НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004[422.8+632]

«До захисту допущено»
Завідувач кафедри
_____ І.А. Дичка
«___»_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Моделі та програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією Blockchain»

Виконав:
студент VI курсу, групи КП-71мп
Васін Костянтин Васильович
_____
(підпис)

Науковий керівник:
Доцент кафедри програмного забезпечення
комп'ютерних систем ФПМ КПІ ім. Ігоря Сікорського,
к.т.н., доцент Цуркан В.В.
_____
(підпис)

Рецензент:
Начальник управління супроводження програми
інформатизації та нормативно-методичної діяльності
Департаменту інформатизації МВС України
к.т.н., доцент Дорогий Я.Ю.
_____
(підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Васіну Костянтину Васильовичу

1. Тема дисертації «Моделі та програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією Blockchain», науковий керівник дисертації Цуркан Василь Васильович, к.т.н., доцент, затверджені наказом по університету від «___»_____ 2018 р. №_____

2. Термін подання студентом дисертації «14» грудня 2018 р.

3. Об'єкт дослідження: процес зберігання електронних документів електронних документів з цифровим підписом.

4. Предмет дослідження: програмні засоби зберігання електронних документів з цифровим підписом.

5. Перелік завдань, які потрібно розробити:

- Проаналізувати програмні засоби зберігання електронних документів з цифровим підписом.
- Побудувати концептуальну модель програмного засобу зберігання електронних документів з цифровим підписом.
- Побудувати об'єктно-орієнтовану модель програмного засобу зберігання електронних документів з цифровим підписом.

– Розробити робочий проект програмного засобу зберігання електронних документів з цифровим підписом.
– Розробити стартап-проект програмного засобу зберігання електронних документів з цифровим підписом.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

– діаграма варіантів використання програмного забезпечення;
– діаграма діяльності програмного забезпечення;
– архітектура програмного забезпечення для зберігання електронних документів з цифровим підписом;
– діаграма компонентів програмного забезпечення.

7. Орієнтовний перелік публікацій:

– Програмний засіб зберігання електронних документів з цифровим підписом за технологією блокчейн, XI наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2018-2).

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання «04» жовтня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.10.2017	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2017	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2018	
4.	Робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2018	

5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2018	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2018	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2018	05.11.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	04.12.2018	

Студент К.В. Васін

Науковий керівник дисертації В.В. Цуркан

РЕФЕРАТ

Актуальність теми. На ринку систем електронного документообігу, можна зустріти два різновиди зберігання. До першого виду відносяться готові системи, які встановлюються на сервери відповідного користувача та налаштовуються під його вимоги. В даному випадку єдиним обмеженням є обсяг пам’яті. Іншим, найбільш популярним серед користувачів, видом є хмарні сховища, використання яких обмежується обсягами пам’яті та швидкістю доступу до нього. Дані різновиди є зручними, але не дозволяють надійно зберігати фали. Це обумовлено залежністю від роботи адміністраторів системи. Окрім, впливу людського фактору на організацію файлового сховища, немає можливості зберігання електронних документів з цифровим підписом в єдиному середовищі. Зокрема, відсутня вбудована інтеграція з сервісом накладання цифрового підпису. Тому актуальним завданням є побудова моделей програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн.

Об’єктом дослідження є процес зберігання електронних документів електронних документів з цифровим підписом.

Предметом дослідження є програмне забезпечення для зберігання електронних документів з цифровим підписом.

Мета роботи синтезувати програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн завдяки побудові його моделей.

Методи дослідження.Теоретичною основою дисертаційних досліджень є теорія моделювання процесів. Зокрема, теорія функціонального, процесного моделювання і моделювання потоків даних – для побудови концептуальної моделі програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн; теорія об’єктно-орієнтованого моделювання – для побудови об’єктно-орієнтованої моделі

програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн.

Наукова новизна роботи полягає в одержанні таких результатів:

– уперше побудовано концептуальну модель програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн, використання якої дозволяє формалізування його роботу на рівні функцій, процесів і потоків даних, а також сформувати та обґрунтувати варіанти використання програмного забезпечення;

– уперше побудовано об’єктно-орієнтовану модель програмного забезпечення на основі формалізування його роботи на рівні функцій, процесів і потоків даних, використання якої дозволяє надати нову якість програмному забезпеченню при створенні або вдосконаленні, зокрема, забезпечити його функціональну придатність до зберігання електронних документів з цифровим підписом за технологією блокчейн.

Практична цінність отриманих результатів полягає у доведенні їх до практичного реалізування, а саме:

– розроблення програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн за його об’єктно-орієнтованою моделлю;

– зберігання електронних документів з цифровим підписом за технологією блокчейн завдяки використанню розробленого програмного забезпечення.

Апробація роботи. Результати роботи пройшли апробацію на XI науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг (ПМК-2018-2)».

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п’яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, оцінено сучасне програмне забезпечення для зберігання електронних документів з

цифровим підписом за технологією блокчейн, обґрунтовано актуальність обраного напрямку досліджень, сформульовано мету і завдання дослідження.

У першому розділі проаналізовано програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн стосовно забезпечення безпеки їх зберігання. За результатами такого аналізування показано необхідність створення відповідного програмного забезпечення, сформовано функціональні та не функціональні вимоги для його синтезування.

У другому розділі побудовано концептуальну модель програмного забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн на функціональному, процесному рівнях, рівні потоків даних у графічних нотаціях IDEF0, IDEF3 та DFD. Це дозволило на функціональному рівні формалізувати його роботу набором функцій. Тоді як на процесному рівні та рівні потоків даних описати взаємозв’язки між етапами зберігання електронного документу та обмін даних між ними.

У третьому розділі на основі концептуальної моделі побудовано об’єктно орієнтовану модель програмного забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн у графічній нотації UML. Це дозволило визначити його варіанти використання, логічну та фізичну структури та, як наслідок, синтезувати програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн.

У четвертому розділі визначено характеристики, метрики якості та проведено функціональне тестування програмного забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн. Наведено контрольні приклади тестування основних його варіантів використання. Визначено системні та апаратні вимоги до клієнтської та серверної частини програмного забезпечення. Описано кроки успішного встановлення і наведено інструкцію адміністратору для

користування програмним забезпеченням для зберігання електронних документів з цифровим підписом за технологією blockchain.

У п’ятому розділі визначено ринкові перспективи стартап-проекту програмного забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення та виведення стартап-проекту на ринок.

У висновках проаналізовані отримані результати.

У додатках наведено структуру програмного забезпечення.

Магістерська дисертація виконана на 122 аркушах, містить 2 додатків та посилання на список використаних літературних джерел зі 55 найменувань. У роботі наведено 22 рисунків та 20 таблиць.

Ключові слова: блокчейн, технологія блокчейн, децентралізована система, електронний документ, цифровий підпис, концептуальна модель програмного забезпечення, об’єктно-орієнтована модель програмного забезпечення, програмне забезпечення зберігання електронних документів з цифровим підписом.

ABSTRACT

Relevance of the topic. On the market of electronic document management systems, two kinds of storage can be found. The first type includes ready-made systems that are installed on the servers of the user and customized to his requirements. In this case, the only restriction is the amount of memory. The other, most popular among users, is cloud storage that is limited to memory and speed of access to it. These methods are convenient, but do not provide high reliability of file storage. This is due to the dependence on the work of system administrators. In addition, the influence of the human factor on the organization of the file repository, to the same there is no possibility of storing electronic documents with a digital signature in a single environment, that is, there is no built-in integration with the service of digital signature overlays. Therefore, the storage of electronic documents with digital signature is vital because of the use of the blockchain technology.

Object of research is the process of storing electronic documents of digital documents with a digital signature.

Subject of research is software tools for storing digital documents with a digital signature.

Research objective is synthesizing of software for storing electronic documents with a digital signature on the blockchain technology by constructing its models.

Research methods. The theoretical basis of dissertation research is the theory of process modeling. In particular, the theory of functional, process modeling and data flow modeling – for constructing a conceptual model of software for storing electronic documents with digital signature on technology blockchain; the theory of object-oriented modeling – for building an object-oriented model of software for storing electronic documents with digital signature on technology blockchain.

Scientific novelty of the work is to obtain the following results:

– for the first time a conceptual model of software for storing electronic documents with digital signature on the technology of blockchain was built, the use of which allows to formalize its work at the level of functions, processes and data streams, as well as to form and justify the use of software;

– the object-oriented software model was first built on the basis of formalizing its work at the level of functions, processes and data streams, the use of which allows to provide new quality software when creating or improving, in particular, to ensure their functional suitability for storing digital documents with digital Signature for technology blockchain.

Practical value of the results obtained is to bring them to practical realization, namely:

– development of software for storing electronic documents with digital signature on the blockchain technology for its object-oriented model;

– storing of electronic documents with digital signature on the blockchain technology due to the use of the developed software.

Approbation. The results of the work were tested at the XIth Conference of Masters and Postgraduate Students "Applied Mathematics and Computing (PMK-2018-2).

Structure and content of the thesis. The master's dissertation consists of an introduction, five sections, conclusions and appendices.

The introduction gives a general description of the work, evaluates the modern software for storing electronic documents with digital signature on the technology blockchain, substantiates the relevance of the chosen direction of research, formulated the purpose and objectives of the study.

The first section analyzes the software for storing electronic documents with a digital signature on the technology of blockchain regarding the safety of their storage. According to the results of this analysis, the necessity of creating the corresponding software has been demonstrated, functional and non-

functional requirements for its synthesis have been formed. The second chapter formalizes the process of storing electronic documents, identifies communications and streams between the stages of storage of electronic documents.

In the third section, based on the conceptual model, an object-oriented software model for storing electronic documents with digital signatures on the blockchain technology in the graphical UML notation was built. This allowed us to determine its usage patterns, logical and physical structure, and, consequently, to synthesize software for storing digital documents with the digital signature of the technology blockchain.

The fourth section defines characteristics, quality metrics, and performs functional testing of software for storing electronic documents with a digital signature on the technology of blockchain. Examples of testing their main uses are given. The system and hardware requirements for the client and server part of the software are determined. Describes the steps for successful installation and provides an instruction to the administrator to use the software for storing digital documents with digital signature technology blockchain technology.

The fifth section defines the market prospects of a startup project for storing digital documents with a digital signature on the blockchain technology, the timetable and principles for organizing production, financial analysis and risk analysis, and measures to promote the offer to investors. The stages of designing and launching a startup project on the market are summarized.

The conclusions are analyzed by the obtained results.

The annexes show the structure of the software.

The master's dissertation is performed on 122 sheets, contains 2 appendices and a link to the list of used literary sources from 55 titles. There are 22 drawings and 20 tables in the work.

Keywords: blockchaine, blockchaine technology, decentralized system, electronic document, digital signature, conceptual model of software, object-oriented model of software, digital documents with digital signature storing software.

РЕФЕРАТ

Актуальность темы. На рынке систем электронного документооборота, можно встретить две разновидности хранения. К первому виду относятся готовые системы, которые устанавливаются на серверы соответствующего пользователя и настраиваются под его требования. В данном случае единственным ограничением является объем памяти. Другим, наиболее популярным среди пользователей, видом является облачные хранилища, ограничивается памятью и скоростью доступа к нему. Данные способы удобны, но не дают высокой надежности сохранения фалов. Это обусловлено зависимостью от работы администраторов системы. Кроме, влияния человеческого фактора на организацию файлового хранилища, к этому же отсутствует возможность хранения электронных документов с цифровой подписью в единой среде, то есть отсутствует встроенная интеграция с сервисом наложения цифровой подписи. Поэтому актуальным является хранение с электронных документов с цифровой подписью благодаря использованию технологии blockchain.

Объектом исследования является процесс хранения электронных документов электронных документов с цифровой подписью.

Предметом исследования является программные средства хранения электронных документов с цифровой подписью.

Цель работы является программно реализовать средство хранения электронных документов с цифровой подписью по технологии blockchain благодаря построению его объектно-ориентированной модели.

Методы исследования. Теоретической основой диссертационных исследований является теория моделирования процессов. В частности, теория функционального, процессного моделирования и моделирования потоков данных – для построения концептуальной модели программного обеспечения хранения электронных документов с цифровой подписью по

технологии blockchain; теория объектно-ориентированного моделирования – для построения объектно-ориентированной модели программного обеспечения хранения электронных документов с цифровой подписью по технологии blockchain.

Научная новизна заключается в получении следующих результатов:

– впервые построено концептуальную модель программного обеспечения хранения электронных документов с цифровой подписью по технологии blockchain, использование которой позволяет формализации его работу на уровне функций, процессов и потоков данных, а также сформировать и обосновать варианты использования программного обеспечения;

– впервые построены объектно-ориентированную модель программного обеспечения на основе формализации его работы на уровне функций, процессов и потоков данных, использование которой позволяет предоставить новое качество программному обеспечению при создании или совершенствовании, в частности, обеспечить их функциональную пригодность к хранению электронных документов с цифровой подписью по технологии blockchain.

Практическая ценность полученных результатов заключается в доведении их до практического реализуемость, а именно:

– разработка программного обеспечения хранения электронных документов с цифровой подписью по технологии blockchain по его объектно-ориентированной моделью;

– хранения электронных документов с цифровой подписью по технологии blockchain благодаря использованию разработанного программного обеспечения.

Апробация работы. Результаты работы прошли апробацию на XI научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг (ПМК-2018-2).

Структура и объем работы. Магистерская диссертация состоит из введения, пяти глав, заключения и приложений.

Во введении дано общая характеристика работы, оценены современное программное обеспечение для хранения электронных документов с цифровой подписью по технологии blockchain, обоснована актуальность выбранного направления исследований, сформулированы цели и задачи исследования.

В первом разделе проанализированы программное обеспечение для хранения электронных документов с цифровой подписью по технологии blockchain по безопасности их хранения. По результатам такого анализа показана необходимость создания соответствующего программного обеспечения, сформирован функциональные и нефункциональные требования для его синтезирования.

Во втором разделе построено концептуальную модель программного обеспечения для хранения электронных документов с цифровой подписью по технологии blockchain на функциональном, процессном уровне, уровне потоков данных в графической нотации IDEF0, IDEF3 и DFD. Это позволило на функциональном уровне формализовать его работу набором функций. Тогда как на процессном уровне и уровне потоков данных описать взаимосвязи между этапами хранения электронного документа и обмен данных между ними.

В третьем разделе на основе концептуальной модели построены объектно ориентированную модель программного обеспечения для хранения электронных документов с цифровой подписью по технологии blockchain в графической нотации UML. Это позволило определить его варианты использования, логическую и физическую структуры и, как следствие, синтезировать программное обеспечение для хранения электронных документов с цифровой подписью по технологии blockchain.

В четвертом разделе определены характеристики, метрики качества и проведено функциональное тестирование программного обеспечения для

хранения электронных документов с цифровой подписью по технологии blockchain. Приведены контрольные примеры тестирование основных его вариантов использования. Определены системные и аппаратные требования к клиентской и серверной части программного обеспечения. Описаны шаги успешной установки и приведена инструкция администратору для пользования программным обеспечением для хранения электронных документов с цифровой подписью по технологии blockchain.

В пятом разделе разделе определены рыночные перспективы стартап-проекта программного обеспечения для хранения электронных документов с цифровой подписью по технологии blockchain, график и принципы организации производства, финансовый анализ и анализ рисков и меры по продвижению предложения для инвесторов. Обзор этапы разработки и вывода стартап-проекта на рынок.

В выводах проанализированы полученные результаты.

В приложениях приведена структура программного обеспечения.

Магистерская диссертация выполнена на 81 листах, содержит 3 приложений и ссылки на список использованных литературных источников из 55 наименований. В работе приведены 20 рисунков и 20 таблиц.

Ключевые слова: блокчейн, технология блокчейн, децентрализованная система, электронный документ, цифровая подпись, концептуальная модель программного обеспечения, объєктно-ориентированая модель програмного обеспечения, программное обеспечение хранения электронных документов с цыфровой подписью.

Аналізування процесу зберігання

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Блокчейн (англ. Blockchain або block chain) – вибудувана за певними правилами безперервна послідовна ланцюжок блоків (зв'язний список), що містять інформацію. Найчастіше копії ланцюжків блоків зберігаються на безлічі різних комп'ютерів незалежно один від одного [1].

Децентралізація – процес перерозподілу, розсіювання функцій, сил, влади, людей або речей від центрального розташування або керуючого органу. Поняття децентралізації використовується в описі групової динаміки, в менеджменті, політичній науці, праві, державному і громадському управлінні, економіці і технології [2].

Хмарне сховище (англ. cloud storage) – являє собою модель схову даних, де цифрові дані зберігаються в логічні пули, а фізичне зберігання охоплює кілька серверів (і часто на різних місцях (локаціях)), фізичне середовище, як правило, належить хостинговим компаніям, вони ж керують цим середовищем. Ці постачальники хмарних систем схову даних відповідають за схов наявної інформації й доступ до неї, та за роботу фізичного середовища. Користувачі купують у постачальників послуг хмарного сховища змогу зберігати там дані [3].

Хешування (англ. Hashing) – перетворення масиву вхідних даних довільної довжини в (вихідну) бітову рядок встановленої довжини, що виконується певним алгоритмом. Функція, яка втілює алгоритм і виконує перетворення, називається «хеш-функцією» або «функцією згортки» [4].

Вузол завантаження, також відомий як хост зустрічі – це вузол у мережі, який надає первинну інформацію про конфігурацію для нових вхідних вузлів, щоб вони могли успішно приєднуватися до мережі. Вузли завантаження переважно знаходяться в децентралізованих однорангових мережах (P2P) через динамічно змінювані ідентичності та конфігурації вузлів учасників у цих мережах [5].

ВСТУП

У сучасних умовах досить важко уявити випадок, щоб у підприємства чи людини не було власного сховища для електронних документів. Створення програмного забезпечення для управління власними файлами є досить затратним, в силу цієї обставин люди почали використовувати готові рішення для організації документообігу.

На ринку систем електронного документообігу, можна зустріти два різновиди зберігання. До першого виду відносяться готові системи, які встановлюються на сервери відповідного користувача та налаштовуються під його вимоги. В даному випадку єдиним обмеженням є обсяг пам’яті. Іншим, найбільш популярним серед користувачів, видом є хмарні сховища, що обмежується пам’яттю і швидкістю доступу до нього.

Дані способи є зручними, але не дають високої надійності збереження фалів. Це обумовлено залежністю від роботи адміністраторів системи. Окрім, впливу людського фактору на організацію файлового сховища, до цього ж відсутня можливість зберігання електронних документів з цифровим підписом в єдиному середовищі, тобто відсутня вбудована інтеграція з сервісом накладання цифрового підпису. Тому актуальним є зберігання з електронних документів з цифровим підписом завдяки використанню технології блокчейн. Для синтезування програмного засобу збереження електронних документів з цифровим підписом пропонується використання технології блокчейн у децентралізованій системі.

Використання даного підходу забезпечує розподілення контролю за збереженням електронних документів. Цей підхід характеризуються високою стійкістю до зломів даних учасників або атак на мережу в цілому. Тому велика кількість документів, які надходять, будуть оброблені.

Основні переваги використання технології блокчейну – це прозорість проведених транзакцій і множинне копіювання всіх цих транзакцій таким

чином, що у кожного учасника процесу завжди є інформація про кожен крок усіх партнерів. Поняття транзакції у контексті блокчейн технології розуміється як запит на збереження одиниці інформації, тобто у конкретному випадку – файлу або електронного документу з цифровим підписом.

Таким чином, на сьогоднішній день проблему безпеки збереження електронних документів з цифровим підписом можливо вирішити за допомогою технології блокчейн. Тому моделювання програмного забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн є актуальним.

1 ПРОБЛЕМАТИКА СИНТЕЗУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ЕЛЕКТРОННИХ ДОКУМЕНТІВ З ЦИФРОВИМ ПІДПИСОМ

1.1. Аналізування процесу зберігання електронних документів

Позаофісне зберігання документів – це передача функцій зі зберігання та обслуговування документів організації професійної архівної компанії. На позаофісне зберігання передаються будь-які документи, терміни зберігання яких не закінчилися: бухгалтерські, банківські кадрові, медичні, включаючи документи, що містять конфіденційну інформацію. Поряд з паперовими документами, зберіганню підлягають аудіовізуальні документи і документи на електронних носіях (CD-диски, флеш-накопичувачі, пристрої пам’яті) [6].

Процес зберігання документів залежить від самого програмного забезпечення. Якщо документ зберігається у хмарному сховищі, то процес починається із завантаження файлу на клієнтську частину. Зазвичай доступ до інтерфейсу клієнтської частини здійснюється за допомогою веб-браузера або з мобільного додатку. Потім надходить запит на збереження документа, де відбувається його перевірка та збереження у сховище даних. Якщо документ зберігається у локальному сховищі, то процес зводиться до переміщення бажаного файлу в потрібну папку.

Процес зберігання документа з електронним цифровим підписом (наприклад на основі алгоритмів симетричного шифрування, який передбачає наявність у системі третьої особи - арбітра, що користується довірою обох сторін або на основі алгоритмів асиметричного шифрування, які на даний момент найбільш поширені [7]) дещо відрізняється від типового збереження тим, що до об’єкта самого файлу додається і його підписана версія, у відповідному форматі (наприклад файли з *.p7s, які використовуються в програмах документообігу для відправки або отримання, або просто переглядання захищених файлів або електронного

листа, зашифрованих за допомогою цифрових підписів) [8]. Класичний процес зберігання документа розширюється функціональністю для роботи з підписом як на клієнті, так і на серверній частинах. На клієнті використовується стороннє програмне забезпечення , який надає можливість накласти підпис, даний процес заключається у надсиланні бажаного файлу до сервісу та підписанням сертифікатом користувача. На вихід отримуємо підпис.

Розлягаючи процес зберігання електронних документів з цифровим підписом, доцільно починати із складових частин, які входять до цієї системи [9]:

– клієнтська частина, забезпечує кінцевому користувачу зрозумілий інтерфейс взаємодії із програмним забезпеченням зберігання документів;

– підсистема аналізу документів. Її завданням є аналіз документа клієнта перед збереженням у сховище;

– система збереження, яка забезпечує процес завантаження даних користувача у зручному вигляді програмного компоненту для збереження даних; забезпечує можливість відтворення документа та повернення на клієнтську частину у зручному вигляді для користувача.

Для клієнтської частини обмеження вносяться, зазвичай зі сторони кінцевого користувача, його побажання із використанням традиційних практик розроблення інтерфейсу. Такими обмеження можуть бути [10]:

– доступність системи у цілодобовому режимі, маєтеся на увазі, що незалежно від місця перебування та часу доби користувач повинен мати змогу отримати свої документи;

– захищеність, користувач повинен бути впевнений, що данні які передаються мережею не можуть потрапити до сторонніх осіб;

– швидкість отримання документів, дана особливість також відносяться і на швидкість мережі, але із застосуванням сучасних

серверних систем та правильно спроектованого програмного забезпечення можна зменшити до прийнятного часу;

– Можливість маніпулювання електронними документами без досвіду роботи. Інтерфейс може реалізуватися як веб-застосування, так і мобільний або десктопний застосунок.

– Тривалість зберігання фалів різний, оскільки якщо файл має юридичну силу, то відповідно і має свій час актуальної дії.

На рис. 1.1 показано узагальнену архітектуру програмне забезпечення для зберігання електронних документів.

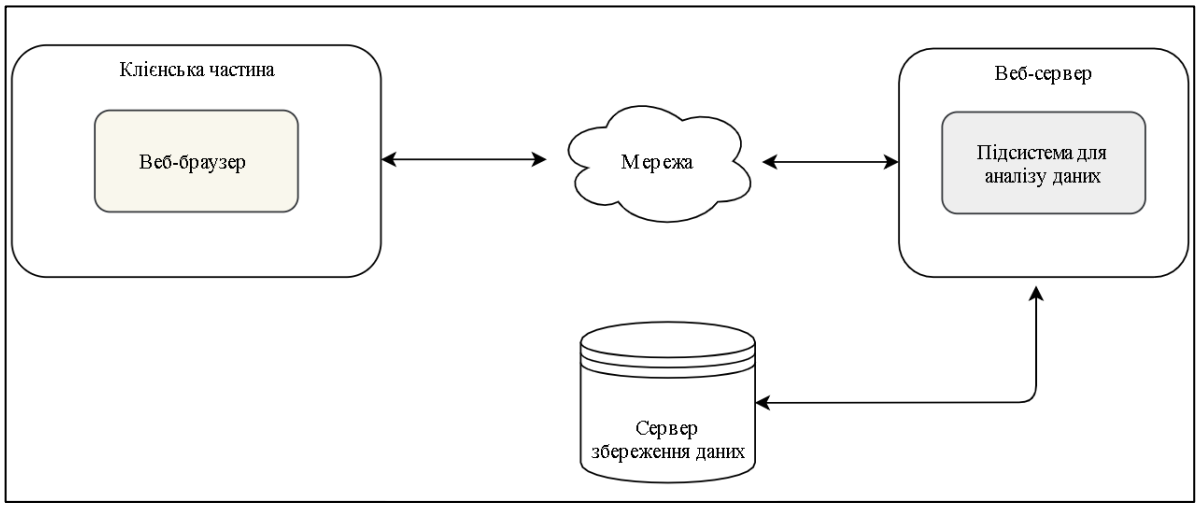


Рис.1.1. Узагальнена архітектура програмного забезпечення для збереження електронних документів

Водночас для розроблення програмного забезпечення зберігання електронних документів з цифровим підписом варто описати особливості означеного процесу та сфери його застосування.

Особливістю сфери застосування програмного забезпечення зберігання електронних документів з цифровим підписом [11]:

– обмеженість обсягу пам’яті засобів накопичення інформації користувачів;
– необхідність зберігання електронного документу у первозданному вигляді для відтворення підпису у майбутньому;

– забезпечення можливості накласти підпис на електронний документ;
– необхідність підтримання доступу до програмного забезпечення у будь-який момент часу з персональних пристроїв користувача;
– надійність зберігання без ризиків втрати даних у разі кібер-атак чи пошкодження носіїв інформації.

Таким вимогам відповідає більшість існуючого програмного забезпечення для зберігання елеронних документів.

Оскільки основним компонентом програмного забезпечення для зберігання електронних документів з цифровим підписом є сховище даних, то доцільно проаналізувати найбільш розповсюджені серед них [12].

На сьогоднішній день сховища електронних документів змінилися, проте застарілі засоби активно використовуються і нині. Це обумовлено тим, що вони не потребують додаткових зусиль від користувача через наявність вбудованих механізмів в операційних системах, наприклад файлові провідники.

Так як вимоги до сховищ електронних документів з цифровим підписом можуть відрізнятися для різних сфер, тому виокремимо найбільш розповсюдженні різновиди сховищ:

– локальний архів, може бути звичайна папка на персональному комп’ютері;
– спеціальний електронний архів, спеціалізовані рішення – системи електронного документообігу (наприклад СЕД, ECM [13]);
– хмарні сховища (наприклад pCloud, MEGA, Google Disk [14]).

З огляду на тенденції розвитку сучасних систем документообігу доцільно зазначити, що основні функціональні розробки стосуються вдосконалення користувацького інтерфейсу, інтерфейсів взаємодії з іншими системами та безпеки даних засобами захисту мережевого доступу до електронних документів [14]. Тому необхідно врахувати зберігання

електронним документів з цифровим підписом та засобів роботи з даними документами, включаючи можливість їх відтворення у майбутньому.

Оскільки існуюче програмне забезпечення досить відрізняються, то доцільно, проаналізувати його для синтезування програмного забезпечення зберігання електронних документів з цифровим підписом.

Таким чином, об’єктом дослідження є процес зберігання електронних документів з цифровим підписом. Тоді як предметом – моделі та програмне забезпечення їх зберігання.

1.2.Аналізування моделей і програмного забезпечення зберігання електронних документів

Локальне сховище

Найбільш доступним місцем для зберігання електронних документів є локальне сховище. При його використанні не доцільно застосовувати спеціальне програмне забезпечення, наприклад. Користувачу достатньо впевнено користуватися операційними системами Windows чи будь-якою Unix подібною. За необхідності забезпечення доступу до електронних документів іншим користувачам можна створити мережеву папку з налаштованим правами доступу, наприклад, тільки для читання. Однак, це не гарантує безпеку файлі, тому надійність доведеться забезпечувати власноруч. Також терміни зберігання слід буде контролювати власноруч, так як місце зазвичай дуже обмежене. Розглядаючи локальне сховище у розрізі програмного забезпечення для зберігання електронних документів з електронним підписом, можна заначити, що дана модель не є придатною для даної цілі, оскільки файл підпису може бути видалений, як і сам оригінал сторонньою особою.

На рис. 1.2 показано узагальнену модель зберігання електронних документів у локальному сховищі.

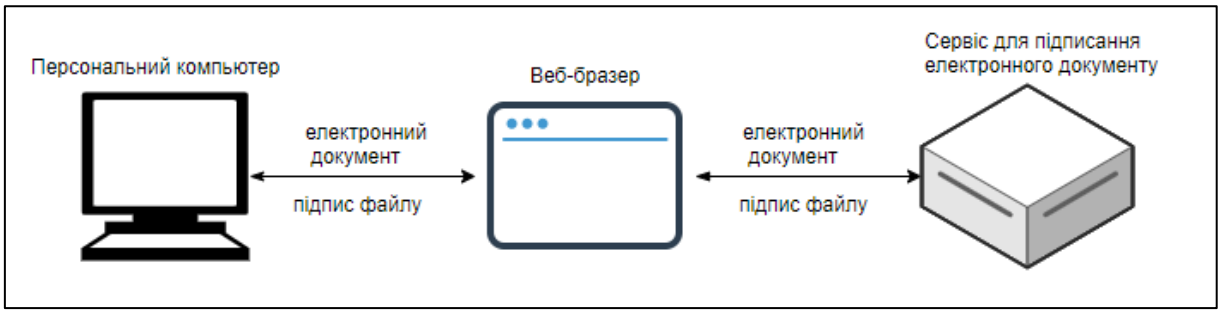


Рис. 1.2. Узагальнена модель зберігання електронних документів у локальному сховищі

Хмарне сховище

Найбільш розповсюдженим та сучасним програмним забезпеченням є зберігання електронних документів у хмарних сховищах [15]. Під хмарними сховищами розуміють модель зберігання електронних документів, де вони зберігаються в логічних пулах, а фізичне зберігання охоплює декілька серверів (і часто на різних місцях (локаціях)). Фізичне середовище, як правило, належить хостинговим компаніям, що керують цим середовищем. Постачальники хмарних систем схову даних відповідають за зберігання і доступ до файлів, та за роботу фізичного середовища [16].

Розглядаючи модель хмарного сховища для електронних документів з електронним підписом, слід відзначити, що процес ділиться на два етапи. На першому етапі користувач повинен підписати файл за допомогою сервісу для підписання електронних документів. Після отримання підпису потрібно надіслати файл у хмарне сховище разом з підписом. Слід відзначити те, що підпис ніяк не асоціюється з самими файлом, що є недоліком так, як при необхідності отримати підпис файлу, користувач власноруч повинен відшукати підпис у виділеному йому сховищі.

На рис. 1.3 показано узагальнену модель зберігання електронних документів у хмарному сховищі.

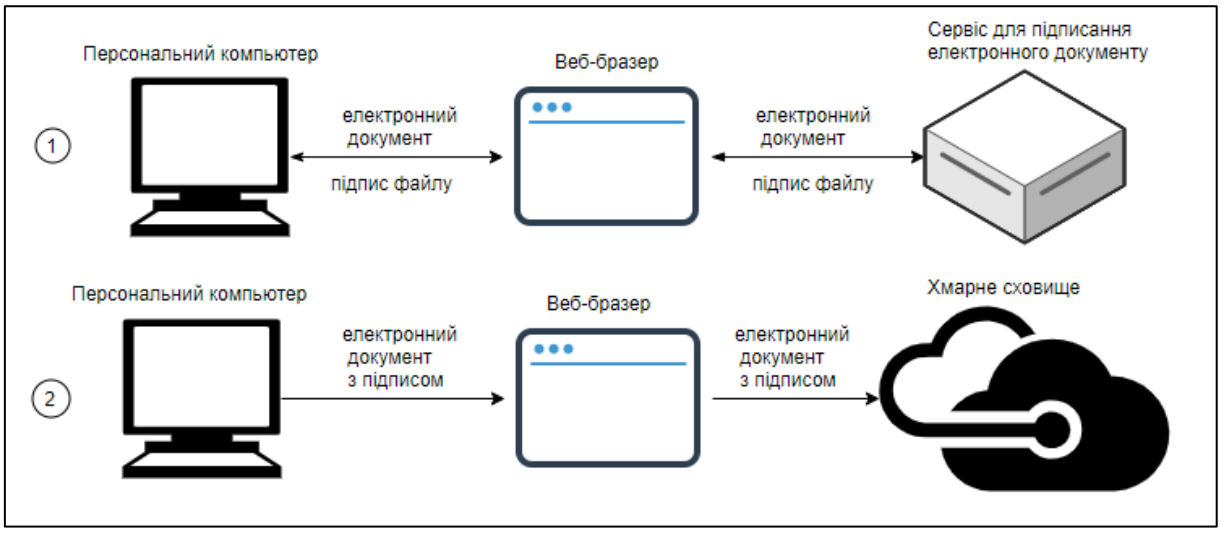


Рис. 1.3. Узагальнена модель зберігання електронних документів у хмарному сховищі

Оскільки однією з найбільш розповсюджених моделей зберігання даних є хмарні сховища [17], то проаналізуємо програмне забезпечення на їх основі.

Google Drive – це файловий хостинг, створений і підтримуваний компанією Google. Його функції включають зберігання файлів в Інтернеті, загальний доступ до них і спільне редагування. До складу Google Диску входять Google документи, таблиці та презентації - набір офісних додатків для спільної роботи над текстовими документами, електронними таблицями, презентаціями, кресленнями, веб-формами та іншими файлами. Загальнодоступні документи на Диску індексуються пошуковими системами [18].

Переваги:

– зручність інтерфейсу та забезпечення доступу з мобільного пристрою та ПК;
– організованість та поширення їх мережею з доступністю за власним бажанням. Крім цього, можливість маніпулювання електронними документами за визначеною роллю, наприклад:

відкрити всім користувачам, тільки по ідентифікатору або за посиланням.

– відображеність тридцять типів файлів у власному інтерфейсі.

Недоліки:

– відсутня можливість синхронізувати папки за межами каталогу Google Drive;

– відсутнє програмне забезпечення для Linux.

Microsoft OneDrive – це файловий хостинг, що надається компанією Майкрософт як частина набору онлайн-послуг. Він дозволяє користувачам зберігати файли, а також інші особисті дані, такі як налаштування Windows або ключі відновлення BitLocker у хмарі [19].

Переваги:

– головною особливістю системи є те, що вона інтегрований у найпоширенішу операційну систему Windows, та власне в офісні документи, що створює не підтримує синхронізацію даних файлів із власним сховищем в реальному часі;

– можливість відновити видалені файли;

– можливість поділитися файлами і папками;

– можливість редагувати офісні документи без встановлення Microsoft Office на комп'ютер.

Недоліки:

– відсутня можливість прегляду історії зміни файлу;

– відсутня можливість синхронізувати папки за межами каталогу SkyDrive;

– відсутнє програмне забезпечення під Linux.

Dropbox дозволяє користувачам створити спеціальну папку на своїх комп'ютерах, яку Dropbox синхронізує таким чином, що вона має однаковий вміст незалежно від того, який пристрій використовується для

перегляду. Файли, розміщені в цій папці, також доступні через веб-сайт Dropbox і мобільні додатки [20].

Переваги:

- використання 256-бітове шифрування AES і шифрування SSL;
- реалізування найкращої в своєму класі технологія синхронізації;
- інтегрування з Microsoft Office 365;
- посилання доступу до файлів за паролем і терміном дії;
- настроюванні рівні доступу.

Mega – файлообмінник, який шифрує весь контент прямо в браузері за допомогою алгоритму AES. Користувачі можуть передавати один одному файли в зашифрованому вигляді, при цьому всі дані зберігаються в «хмарі». Ключі доступу до файлів не публікуються у відкритому доступі, а поширюються по схемі Friend-to-Friend, між довіряють один одному користувачами [21].

Переваги:

- інтегрування чату;
- щифрує весь контент прямо в браузері за допомогою алгоритму AES; користувачі можуть передавати один одному файли в зашифрованому вигляді, при цьому всі дані зберігаються в «хмарі»; ключі доступу до файлів не публікуються у відкритому доступі, а поширюються по схемі Friend-to-Friend, між довіряють один одному користувачами.

Недоліки:

- не висока швидкість доступу.

Для групи людей зручно використовувати системи, які б користувачі могли розгорнути на власних серверах і вже самостійно настроювати доступ та адмініструвати їх. Дані рішення не відносяться до бюджетних варіантів, що є відповідно мінусом. Прикладом такої системи може служити Openkm.

– дана система дозволяє збирати інформацію з будь-якого цифрового джерела;
– мати спільні документи із колегами;
– керувати цифровим підписом.

Решта функціоналу може співпадати з іншими, що стосується керуванням документами.

Наступним сервісом який компанії можуть використовувати для власного документообігу це HotDocs. Даний сервіс є лідером на світовому ринку, але відсутній функціонал із роботою електронно-цифрового підпису.

Слід також виокремити Confluence. Дана система є відкритою, що допомагає створювати, керувати та співпрацювати з будь-яким вмістом від планів запуску продукту до маркетингових кампаній. Легко працювати в спеціалізованих та організованих просторах, підключатися до різних команд, а також інтегрувати їх із набором Atlassian.[22]

Для наочного представлення результатів аналізування програмного забезпечення зберігання електронних документів з цифровим підписом побудуємо таблицю їх порівняння (див. табл. 1.1). Водночас зведені в табл. 1.1 засоби використовуватимуться як прототипи при синтезування програмного забезпечення зберігання електронних документів з цифровим підписом.

Таблиця 1.1

Результати порівняння програмного забезпечення зберігання електронних документів з цифровим підписом

Програмне забезпечення / Критерій порівняння	Google Disk	Mega	Atlassian	Confluence
Зберігання інформації без можливості модифікації	–	–	–	–

Продовження табл. 1.1

Можливість накладання електронного підпису	–	–	–	–
Можливість зберігання електронного документу з електронним підписом	±	±	±	±
Захист від несанкціонованого доступу до фалів	+	+	+	–

Таким чином, актуальним завданням є побудова моделей програмного забезпечення зберігання електронних документів з цифровим підписом для його синтезування.

1.3.Синтезування програмного забезпечення зберігання електронних документів

Застосування існуючих методів та програмних засобів у сфері документообігу дозволяє створювати доволі потужні та цілком робочі рішення. Але не всі рішення забезпечують стовідсоткову захищеність даних до змін та у разі створення відкритих баз документообігу вони можуть бути зміненні під впливом людини, яка адмініструє усі данні, що в свою чергу є недоліком, який перекривається зручним користувацьким інтерфейсом та в деяких рішеннях криптографічними операціями над даними.

Для вирішення поставленої проблеми у сфері документообігу пропонується використання технології блокчейн у децентралізованій системі.

Децентралізована система є підмножиною розподіленої системи. Розподілений контроль, або контроль, в якому кожен компонент системи однаково відповідає за участь у глобальній, складній поведінці, діючи на

місцевій інформації відповідним чином. Децентралізовані системи характеризуються дуже високою стійкістю до зломів даних учасників або атакам на мережу в цілому, тому все частіше стають базовими для фінансових мереж і платформ. Немає якогось одного загального "командного центру", зламавши який вийде знищити всі дані про угоду та її учасників або підмінити їх.

На рис. 1.4 зображено типи систем для предметного розуміння майбутньої системи.

Рис. 1.4. Види систем

Для забезпечення біль надійного збереження даних від навмисних змін, пропонується використати технологію блокчейн у децентралізованій системі.

Головні переваги використання блокчейна - це прозорість проведених транзакцій і множинне копіювання всіх цих транзакцій таким чином, що у кожного учасника процесу завжди є інформація про кожен крок всіх партнерів. Це забезпечує належний рівень відкритості угоди - весь ланцюжок транзакцій дублюється і зберігається в незмінному зашифрованому вигляді у кожного учасника, не вийде якось підробити її. Під учасниками розуміють сервери, на яких розвернуте програмне забезпечення майбутньої системи. Поняття транзакції у розрізі блокчейн технології розуміється як запит до системи для збереження одиниці інформації, тобто у конкретному випадку – файлу [23].

Програмний засіб має на меті програмну реалізацію засобу зберігання електронних документів з цифровим підписом за технологією блокчейн завдяки побудові його об’єктно-орієнтованої моделі.

Для реалізації даного методу зберігання документів було вирішено використовувати документи із накладем електронно-цифровим підписом, відповідно до вимог до юридичної сили документа в рамках законодавства України, посилаючись на «ПОЛОЖЕННЯ про застосування електронного підпису», з якого слідуює, що електронний документ набуває юридичної сили після накладання електронно-цифрового підпису [24]. Використовуючи особливість електронного підпису буде можливо стовідсотково ідентифікувати власника документа. Також даний тип підпису неможливо підробити.

Виходячи з мети та призначення майбутнього сервісу, можна висунути вимоги до реалізації. Данна система повинна реалізовувати систему яка включає сервіс збереження документів, тобто сам блокчейн, сервіс обробки документів перед зберіганням, та інтерфейс за допомогою якого, користувач зможе зробити транзакцію до системи. Відповідно головною вимогою до системи є надійність. Тобто система повинна використовувати безпечні канали передачі даних та реалізовувати авторизацію користувача за стандартом OAuth 2.0, щоб у разі розширення функціоналу іншими сервісами не виникла питань з аунтифікацією іншого сервісу у системі [25].

Так як система реалізує децентралізацію то відповідно для мінімального функціювання системи необхідно два або більше однорангових вузлів, які конструктивно складаються з блокчейну та сервісу перевірки документів.

Для побудови системи стійкої до навантаження використовується вузол навантаження. В рамках децентралізованої системи він буде представляти схожий вузол з іншими, але його мета полягає в тому, щоб надавати нові сполучення вузлів з достатньою кількістю конфігураційних

даних, щоб новий вузол потім успішно об'єднував мережу та доступ до ресурсів, в рамках даної системи це - блокчейн.

Коли новий вузол запускається або надходить в мережу, після підключення до мережі, перше, що він повинен зробити, це оновити базу даних. Перший блок, який він має, - це блок «генерації», який є першим вузлом в блок-схемі та вбудований у програмне забезпечення клієнта. Коли новий вузол надсилає повідомлення про версію, він також включає в себе «висоту» своєї бази даних блоків (кількість блоків). Приймаючий вузол буде бачити висоту і порівняти його з власною копією блочного шаблону. Якщо кількість блоків, які отримує одержувач, перевищує відправника, то він буде точно знати, скільки вузлів не вистачає відправнику. Він може відправити відсутні блоки, щоб «наздогнати» останню версію блокчейна [26].

На рис. 1.5 можна побачити загальну архітектуру засобу зберігання електронних документів з ЕЦП за технологією блокчейн.

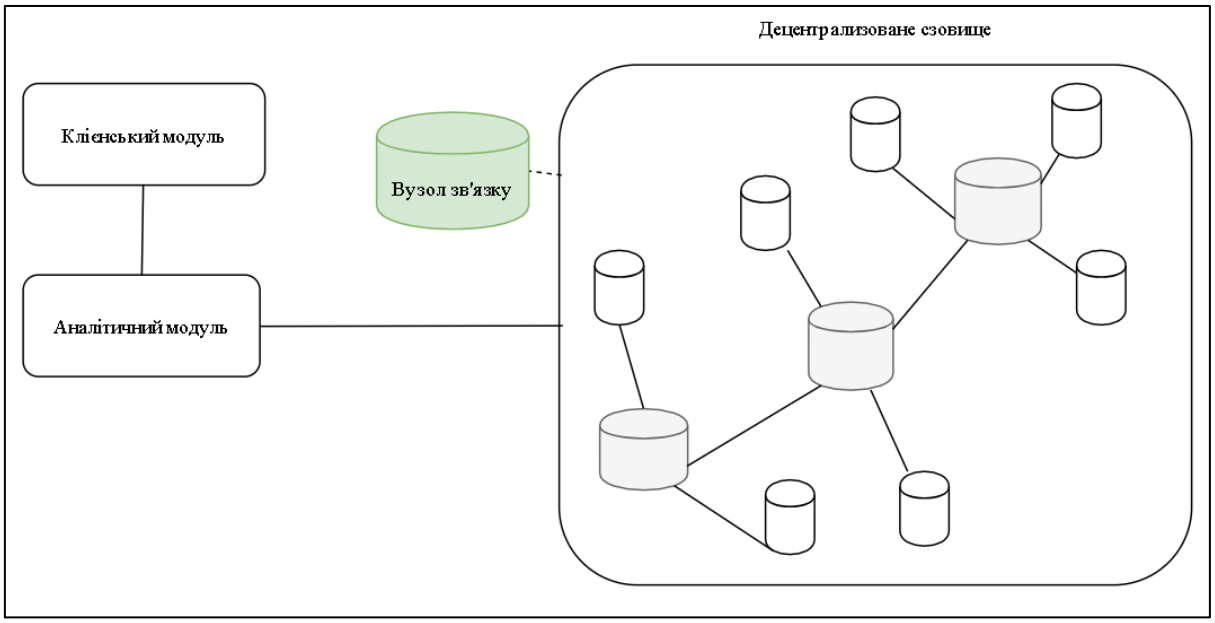


Рис. 1.5. Загальна архітектура майбутньої системи зберігання електронних документів з ЕЦП за технологією блокчейн

Для обміну даними в мережі буде використовуватися безпечне з’єднання засобом використання протоколу шифрування SSL. Даний вид увійшов у стандарт для передачі даних мережею, особливо при роботі з приватними даними користувача, при передачі третій системі.

Для підтримки роботи всієї системи та повинно бути розгорнуто як найменш три сервери, два з яких будуть представляти вузли децентралізованого сховища, які будуть зберігати блокчейн, та третій це вузол зв’язку. Сервери потрібно розгортати на операційній системі Windows, так як розроблена система буде реалізовано за допомогою технологій представлені для .NET платформи.

Необхідне представлення зав’язків між об’єктами програмного забезпечення для коректної побудови архітектури, слід побудувати концептуальну модель у нотації UML. Для уникнення потенційної помилки з втратою даних необхідно побудувати діаграму потоків даних у нотації DFD. Побудовані діаграми вирішують проблеми розуміння контексту та інформаційної наповненості програмного забезпечення, тому для вирішення проблеми об’єктної архітектурі потрібно побудувати об’єктно-орієнтовану модель за нотацією UML.

Таким чином, для синтезування програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн необхідно:

1) побудувати концептуальну модель зберігання електронних документів з цифровим підписом за технологією блокчейн у нотації IDEF;

2) побудувати об’єктно-орієнтовану модель зберігання електронних документів з цифровим підписом за технологією блокчейн у нотації UML;

3) розробити програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн.

1.4.Висновки

Існуючі підходи до збереження електронних документів дозволяють отримувати доступ до власних документів у будь-який час. З функціями обмеження доступу по ролям, редагуванням та у випадках власних систем документообігу не обмеженій обсяг пам’яті. Не вирішеним питання залишаться забезпечення стовідсоткового захисту від видалення та спотворення даних користувачів, так як адміністрування системи залишаться за людиною, а не надання його права самій системі та отримання функціоналу для роботи з ЕЦП для документів, та власне їх зберігання з накладеним ЕЦП у сторонній системі. Для вирішення даної проблеми було проаналізовано існуючі рішення на ринку систем. Визначанню, на які складові в більшості випадків розгортається класичні системи документообігу. Проаналізовано види доступу до власних документів та надійність зберігання власних документів у персональних та сторонніх системах.

Серед найбільш популярних систем було обрано декілька прототипів різних типів зберігання документів. Хмарні сховища виявилися найбільш прогресивними з точки зору користувача з не великими потребами в обсязі пам’яті. Також найчастіше доступ до власних документів можна здійснити через всі можливі пристрої, у вигляді як веб-застосунку так і у мобільному додатку. Локальні системи документообігу потребують більше ресурсу, як людського так і матеріального. В більшості випадків можливий доступ тільки через десктопні застосунки. Тому даний вид рідко зустрічається у власному користуванні, більшість випадків приходить на організацію сховища документів для компаній. Жодна з представлених систем на реалізує функціоналу зберігся документів з накладеним ЕЦП у самій системі.

Узагальнюючи всі проаналізовані методи зберігання документів та існуючі рішення на ринку було синтезовано програмний засіб, який базується на блокчейні у децентралізованій системі. Даний спосіб

підвищує надійність збереження документів, у тому вигляді, в якому вони потрапили до сховища, та мінімізую втручання адміністратора у систему окрім випадків розгортання нових вузлів системи, та налаштування вузла навантаження, тобто за функціювання системи документообігу, використовуючи дану технологію, буде відповідати сама мережа. Також синтезовано функціональність для робити за підписаними ЕЦП документами.

Основні результати розділу опубліковано в [27].

2 КОНЦЕПТУАЛЬНА МОДЕЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ЕЛЕКТРОННИХ ДОКУМЕНТІВ З ЦИФРОВИМ ПІДПИСОМ

2.1.Формалізування процесу зберігання електронних документів

Для створення системи зберігання документів необхідно виділити ключові етапи, які завершуються збереженим документом у децентралізованому сховищі, використовуючи технологію блокчейн.

Для наочного зображення ключових процесів, їх будо промодельовано за допомогою контекстних діаграм (див. рис. 2.1) [27].

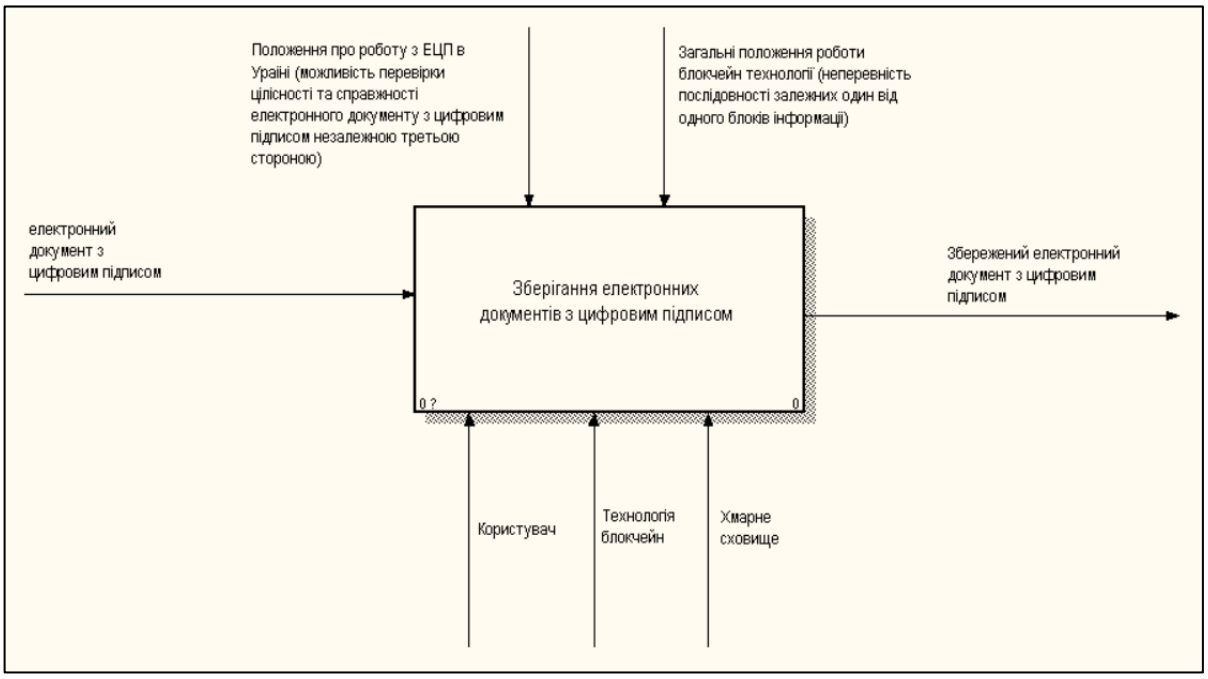


Рис. 2.1. Функціональна модель програмного забезпечення

Функціональна модель зображує найбільш абстрактний опис програмної системи. Вхідними даними від користувача завжди є підписаний документ, в результаті проходження всіх процесів, результатом є підписана версія файлу збереженого до сховища. Система керується обмеженнями, які висуває система до вхідних даних та загальними принципами організації сховища за технологією блокчейн. Виходячи с

того, що процес збереження не обходиться без накладання підпису, слід зауважити, що обраний сервіс для підписання повинен відповідати положенню про накладання ЕЦП, а саме можливість підтвердити її дійсності. Підписані данні є повністю юридично значимі, а підписанта однозначно можна ідентифікувати так як інформація про сертифікат також зберігається у підписі. Механізмами в даному процесі виступають користувач та хмарне сховище. Роль користувача полягає у надходженні документу із цифровим підписом та ініцілізування процесу зберігання документу. Хмарне сховище є ресурсом так як даний процес реалізує собою модель схову даних, де цифрові дані зберігаються в логічні пули, а фізичне зберігання охоплює кілька серверів [27, 28].

Діаграма А0 декомпозуються на чотири основні процеси, виконання яких є обов'язковим продовж усього процесу до збереження (див. рис. 2.2).

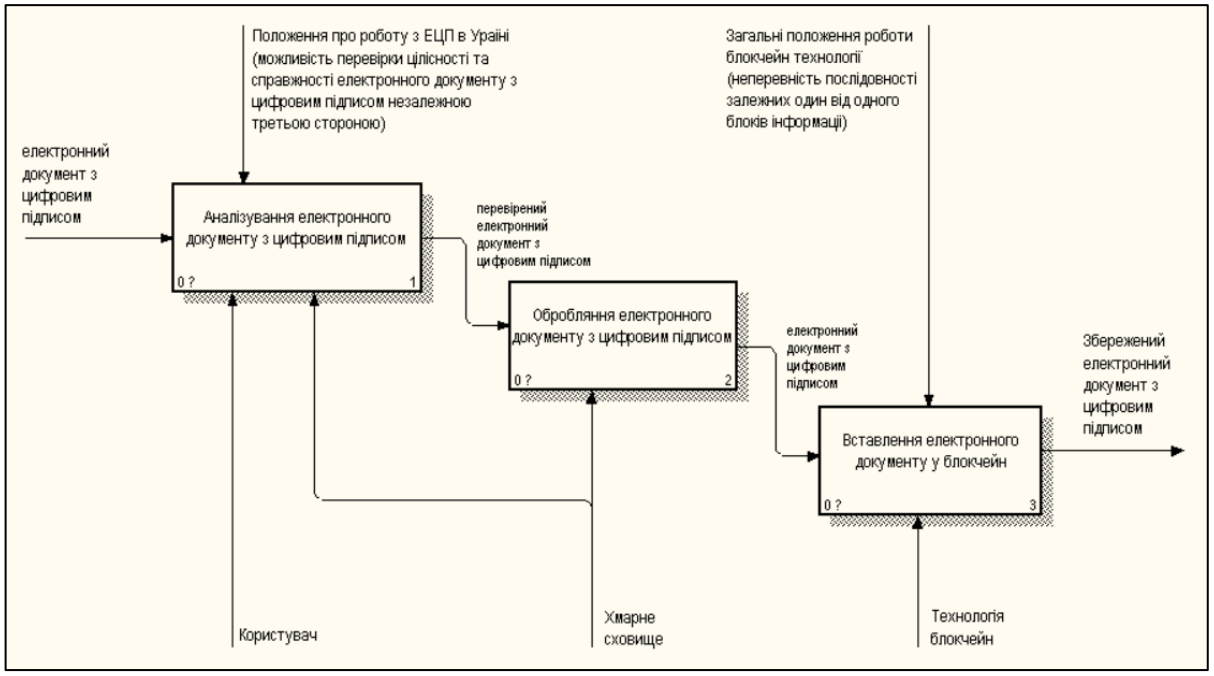


Рис. 2.2. Декомпозиція функціональної моделі програмного забезпечення

Першим етапом є аналізування вхідних даних, яке полягає в фільтрації некоректних запитів на збереження електронного документу з цифровим підписом. Перевірка даних складається з декількох етапів. Першим етапом

є перевірка моделі даних, що приходить у запиті, вона повинна складатися з:

- посилання на отримання підпису документа. За допомогою даного посилання визначається шлях запиту на сервер сервісу, де був підписаний документ, та у разі його наявності продовжується перевірка запиту на збереження документу;
- електронний документ, який повинен бути у форматі послідовності байтів;
- назва електронного документа, даний параметре не є обов'язковим тому, що у випадку його відсутності, назва буде отримана з електронного документа з отриманої моделі;
- ідентифікатор користувача, дане поле моделі повинно бути представлене у вигляді статистично унікального 128-бітний ідентифікатора, який забезпечує майже стовідсоткову унікальність користувача у системі [29];
- дата створення запиту, дана інформація слугує для того, що була можливість відфільтрувати данні за датою отримання файлу.

Якщо електронний документ не відповідає вимогам до моделі, то припиняється процес зберігання документу та повертається відповідь про причину зупинки процесу. У випадку успішного проходження перевірки електронний документ з цифровим підписом продовжує перевірку.

Наступним етапом є отримання файлу з підписом до отриманого документу. Формується запит за отриманою адресою з додаванням параметра авторизації у заголовок. Якщо у відповідь надходить файл, тоді конкатенується у модель, яка є агрегатом усіх необхідних даний для створення блоку.

Результатом даної обробки є перетворений електронний документа у байтовому форматі до строкового, отримання файлу підпис, створення структур даних, які на наступному етапі будуть складати блок.

Найважливішим етапом процесу зберігання електронних документів з цифровим підписом за технологією блокчейн є встановлення електронного документу у неперервний ланцюжок блоків, які складаються з документі та допоміжної інформації. Останній етап на діаграмі декомпозиції отримує структуру даних з інформацією, якої достатньо доля створення блоку (див. рис. 2.3) [27].

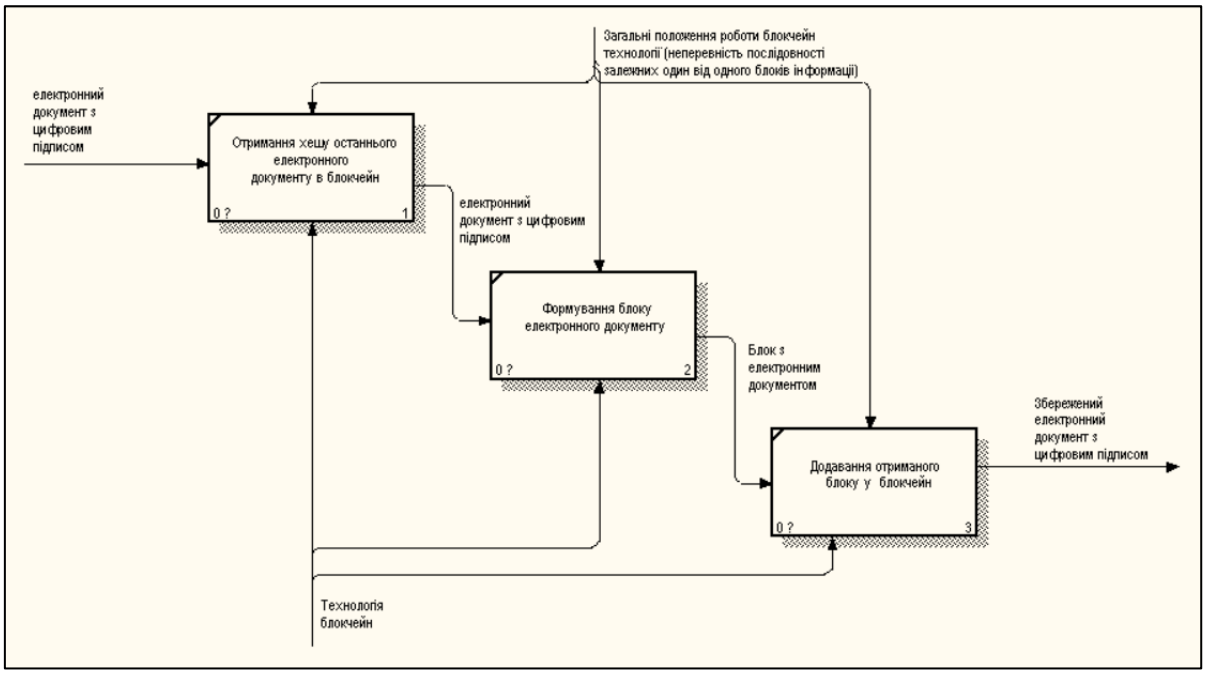


Рис. 2.3 Декомпозиція функціональної моделі програмного забезпечення

Процес встановлення електронного документу у блокчейн починається з отримання останнього блоку з блокчейну. Далі створюється хеш блоку та записується у його заголовок, який є відповідно посиланням, що в результаті буде створювати неперервний ланцюжок блоків. Після успішної операції отримання хешу блоку та формування наступного блоку, відбувається збереження та повернення відповіді з результатом проходження процесу [27].

Даний процес відбувається на всіх вузлах децентралізованої системи. Коли вузол отримує повідомлення про оновлення версії блокчейну, виконується запит на отримання оновлення локального сховища.

2.2.Визначення зв’язків між етапами зберігання електронних документів

Для упорядкування послідовності подій, які відбуваються у програмному забезпеченні відносно основних процесів, було побудовано діаграму, на якій зображено опис технологічних процесів із вказанням, що відбувається на кожному етапі та опис переходів станів об’єктів, із зазначенням проміжних станів [30].

На рис 2.4 зображено декомпозицію функціональної моделі. Так як основними процесам відповідно до декомпозицій IDEF0 є процеси аналізу вхідного документу та його обробляння то розглянемо їх у нотації IDEF3 [31].

Процес аналізування електронного документу з цифровим підписом починається з перевірки відповідності формату вхідного документу до дозволених форматів. Модель файлу, яка надходить на функцію запиту отримує об’єкт з поточного веб контексту та отримує формат файлу. У випадку надсиланні дозволеного формату процес переходить до іншої функції. Наступний етап заключається в отриманні підпису файлу з моделі, якщо вона наявна, то процес аналізування вхідного документу завершується і переходить то процесу обробки електронного документу.

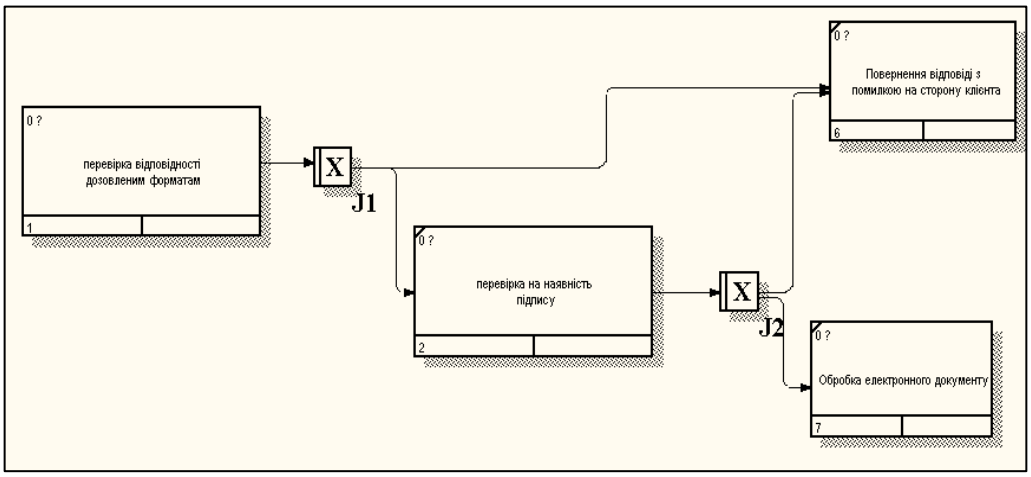


Рис. 2.4 Декомпозиція функціональної моделі в нотації IDEF3

Наступний етап це обробки електронного документу з цифровим підписом. Даний етап починається з отримання адреси випадкового вузла системи зберігання електронних документів.

Мета цього процесу заключається у забезпечення цілковитої безпеки збереження електронних документів та реалізації децентралізованої системи. Отримання випадкового вузла гарантує те, що всі учасники мережі є рівноправними і не відрізняються один від одного. Далі відбувається запит на відправку файлу для збереження у блокчейні [27]. Інші учасники системи оповіщаються про те, що версія блокчейну оновилась, тобто додався ще один блок з інформацією тому необхідно оновити власний ланцюжок блоків. Таким чином з часом данні актуалізуються на всіх вузлах (див. рис.2.5).

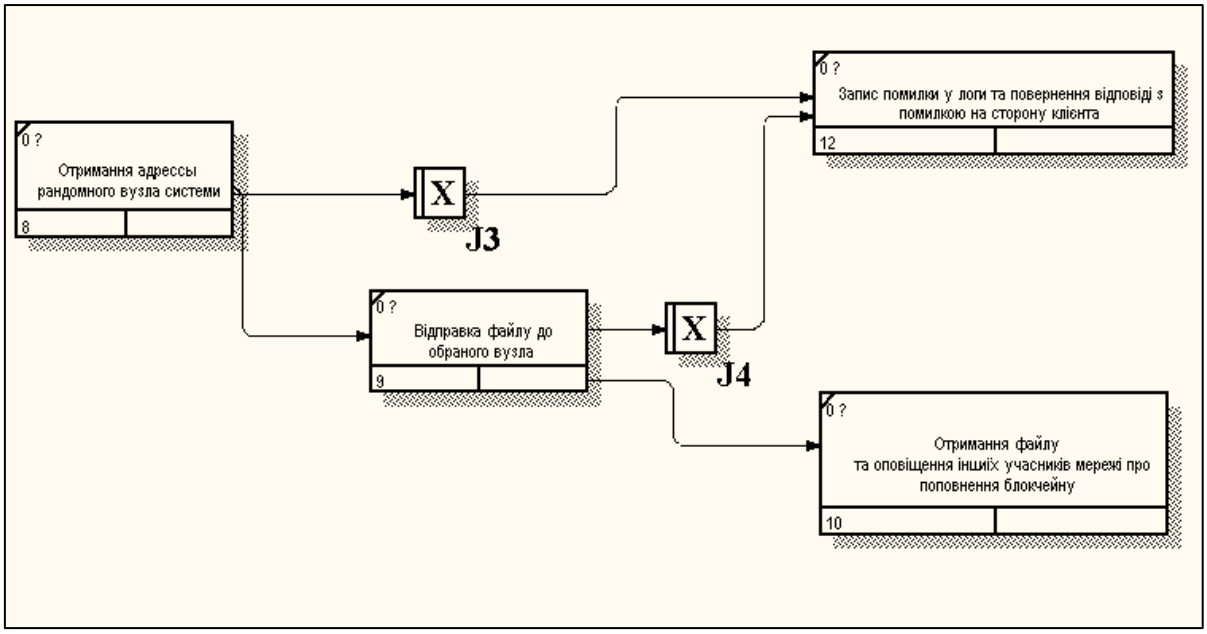


Рис. 2.5 Декомпозиція блоку «Обробка електронного документу з цифровим підписом»

Після проходження етапів аналізування електронного документу та його обробки вузли починають актуалізувати наявну інформацію, шо є початком процесу становлення електронного документу у блокчейн [27].

Після успішної операції встановлення, процес збереження документу рахується завершеним, а данні можливо відтворити за бажання методом відправки запиту на окремий вузол, чи за допомогою клієнтської частини.

2.3.Визначення потоків даних між етапами зберігання електронних документів.

Для визначення потоків даних програмного забезпечення для збереження електронних документів підписом було використано методику графічної структурного аналізу DFD. Даний вибір був обмовлений тим, що система передбачу збереження інформації та виконання операції над ними. Даний вид нотації визначає с чого складається інформаційна складова програмного забезпечення, та які інструменти необхідно для того, щоб її обробити (див. рис. 2.6) [27, 32].

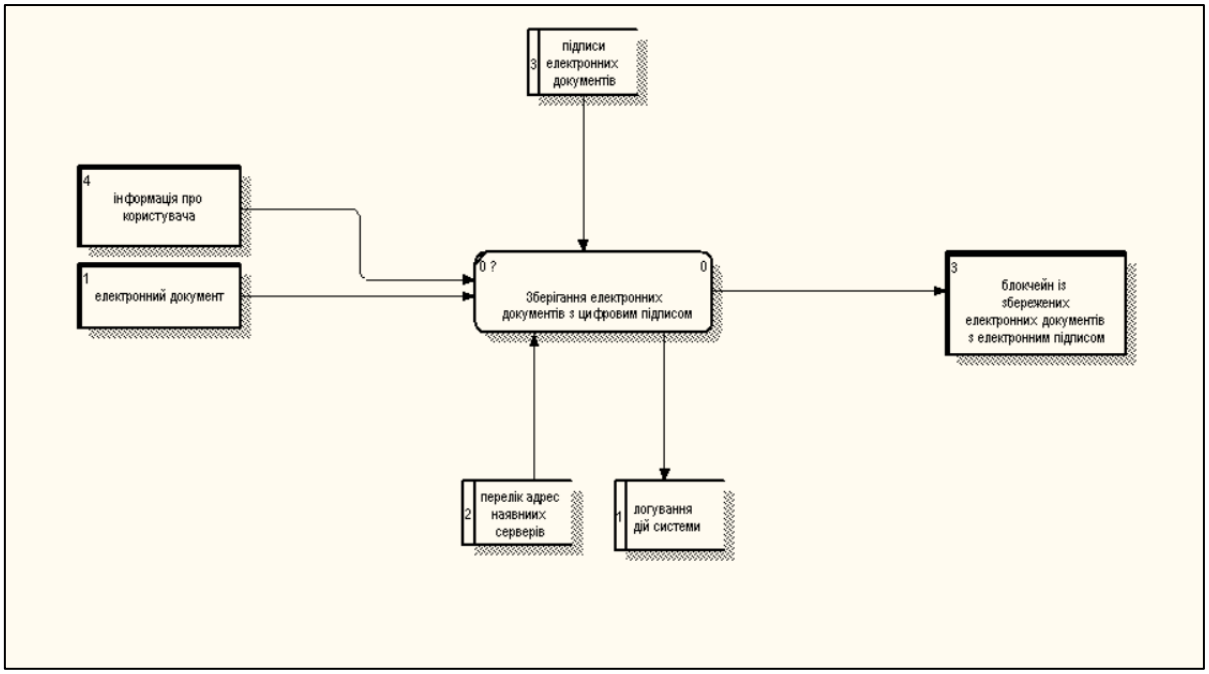


Рис. 2.6 Модель потоків даних програмного забезпечення

На рис. 2.6 зображено декомпозицію діаграми, що зображена на рис. 2.5. Для реалізації програмного забезпечення головним процесом, якого є зберігання електронних документі з цифровим підписом

сформулюємо джерела даних, які потім можна легко інтерпретувати в код програми чи бази даних. Сховищами даних виступають:

- інформація про користувача, вона надходить у заголовку Http-запиту та у моделі із файлом для збереження;
- електронний документ, даний потік представлений Http-запитами, які надсилає користувач;
- підписи електронних документів, являють собою сховище, на стороні сервісу підписання екстрених документів. При необхідності отримати підпис, це можна бути здійснити Http-запитом;
- перелік адрес наявних серверів, являє собою список який зберігається у файлі у вузлі зв'язку і у разі додавання нового вузла від буде поповнюватися;
- легування дій системи служить сховищем, яке реєструє усі дії системи, у разі поликової ситуації можна бути дістати стек подій та дізнатися причину помилки;
- блокчейн із збереженими електронними документами з електронним підписом, являють собою головне сховище куди зберігаються усі документи користувачів.

Головним процесом програмного забезпечення є збереження електронних документів. На вхід поступає інформація про користувача та електронний документ, дана інформація створюється на етапі ініціалізації запиту на збереження електронного документу на клієнтській стороні та приходить разом із запитом. Перелік адрес наявних сервері є сховищем, звідки отримуються інформація про шлях куди відпрядати обробленій документ для збереження у блокчейн [27]. Всі події, які відбуваються під час виконання процесу збереження легуються у сховище «логування дій системи». Підписи електронних документів першочергово створюються у сервісі для підписання, тому данні про підпис запитом затягуються до системи, а сервіс слугує лише їх джерелом у рамках роботи програмного

забезпечення. У результаті проходження всіх внутрішніх процесів збереження електронного документу ми отримуємо неперервний ланцюжок блоків, які складають сховище електронних документів з цифровим підписом.

2.4.Висновки

За допомогою методології функціонального моделювання та графічної нотації IDEF0, було формалізовано та описано процеси, які відбуваються в середині програмного забезпечення для збереження електронних документів з цифровим підписом. Зокрема для визначення основних процесів було декомпозовано IDEF0 на підпроцеси, що в результаті дало змогу виділити окремі функціональності та можливість коректної побудови фізичної та логічної структур програмного забезпечення.

Було визначено зв’язки між процесами використовуючи нотацію IDEF3. Виокремлено окремі процеси проведено декомпозицію та визначено причин та наслідки між внутрішніми процесами. Це дало змогу упорядкувати виклик потрібних функцій при розробці програмного забезпечення та у разі виключних ситуацій правильно прореагувати на них.

Для визначення потоків даних програмного забезпечення було використано діаграму потоків даних. Було описано зовнішні та внутрішні відносини джерел даних та сховищ даних, до яких відбувається доступ. Результатом моделювання потоків даних є визначенні структури даних та моделі, які надходять в тілі запиту.

Результатом моделювання програмного забезпечення для збереження електронних документів за технологією блокчейн є проаналізовані процеси та основа для побудови об’єктно-орієнтованої моделі, що описується у розділі 3.

Основні результати розділу опубліковано в [27].

3 ОБ’ЄКТНО-ОРІЄНТОВАНА МОДЕЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ЕЛЕКТРОННИХ ДОКУМЕНТІВ З ЦИФРОВИМ ПІДПИСОМ

3.1.Специфікування вимог до програмного забезпечення

Для визначення всіх аспектів роботи програмного забезпечення для збереження електронних документів необхідно визначити вимоги, що ставляться до системи. Вимоги ставляться к до функціоналу, доступного користувачеві, так і до можливостей системи. Основні вимоги наведені в табл. 3.1-3.8 та на рис. 3.1 [27].

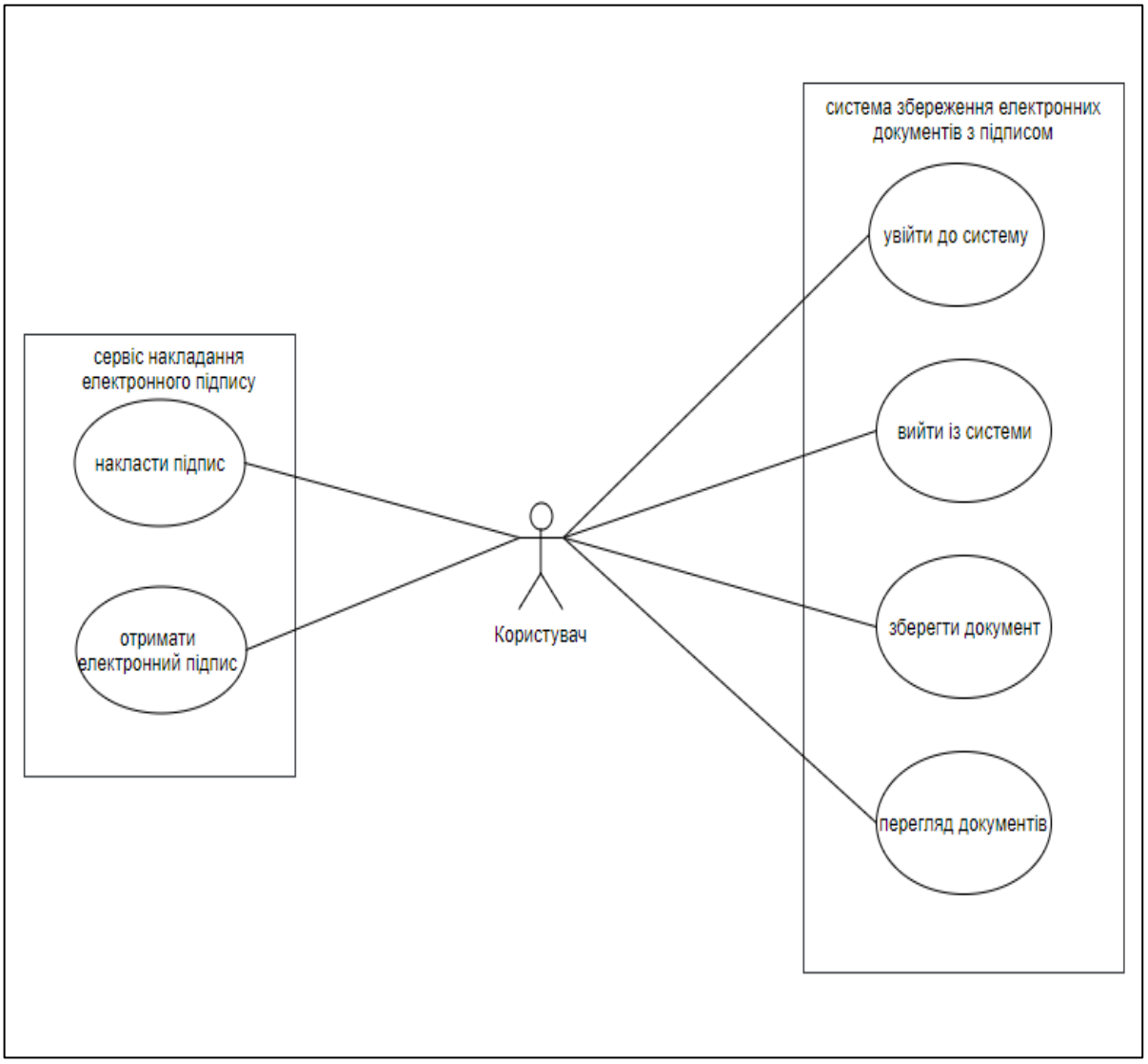


Рис. 3.1. Діаграма варіантів використання програмного забезпечення

Для зображення взаємодії користувача та програмного забезпечення, використовуються діаграми використання, за допомогою яких можна судити чи майбутня система виконує своє призначення і чи зможе вона полегшити користувачеві виконання його роботи взагалі. Дають відповідь на запитання , які з виконуваних функцій важливіші і для кого. Тому використовують опис функціональних вимог через варіанти використання. Розглядаючи класичний сценарій використання розроблюваного програмного забезпечення для збереження електронних документів можна виділити основні кроки, з огляду на діаграму використання:

Крок 1. Користувач входить до системи використовуючи логін, як ідентифікатор.

Крок 2.Завантаження електронного документу за допомогою наданого інтерфейсом інструменту.

Крок 3. Підписання електронного документу за допомогою сервісу накладання електронного підпису.

Крок 4. Відправка підписаного електронного документу для збереження на сервері.

Таблиця 3.1

Варіант використання «Авторизація у системі»

Найменування	Авторизування користувача у системі
Первинний актор	Користувач
Інші актори	Система накладання підпису
Опис	Можливість ідентифікуватися у системі та отримати токен авторизації
Попередні умови	Відкрита сторінка входу до системи
Вихідні умови	Отриманий токен та вхід у систему

Таблиця 3.2

Варіант використання «Вийти із системи»

Найменування	Вийти із системи
Первинний актор	Користувач
Інші актори	Немає
Опис	Можливість завершити поточну сесію та вий з системи
Попередні умови	Бути авторизованим у системі
Вихідні умови	Сесія завершується

Таблиця 3.3

Варіант використання «Підписати електронний документ»

Найменування	Підписання електронного документ аза допомогою стороннього сервісу
Первинний актор	Користувач
Інші актори	Система накладання підпису
Опис	Можливість накладання електронного підпису за допомогою стороннього сервісу
Попередні умови	Відкрита стартова сторінка клієнтської частини
Вихідні умови	Підписаний документ з можливістю отримати файл підпису

Таблиця 3.4

Варіант використання «Зберегти електронний документ з цифровим підписом»

Найменування	Зберегти електронний документ з цифровим підписом
Первинний актор	Користувач
Інші актори	Система накладання підпису
Опис	Можливість зберегти електронний документ з цифровим підписом
Попередні умови	Відкрита сторінка завантаження електронного документа з наявністю підпису.
Вихідні умови	Збережений електронний документ з електронним підписом

Таблиця 3.5

Варіант використання «Обробка запитів на збереження електронних документів»

Найменування	Обробка запитів на збереження електронних документів
Первинний актор	Система
Інші актори	Немає
Опис	Можливість обробити запит на створення файлу, провівши аналіз електронного документа та наявність підпису до нього
Попередні умови	Налаштоване середовище програмного забезпечення та забезпечена можливість отримати доступ то інтерфейсів .

Продовження табл. 3.5

Вихідні умови	Оброблений запит на збереження електронного документу

Таблиця 3.6

Варіант використання «Завантаження файл підпису за його ідентифікатором»

Найменування	Завантаження файл підпису за його ідентифікатором
Первинний актор	Система
Інші актори	Сервіс підпису електронних документів
Опис	Можливість відправити запит на отримання підпису до електронного документу за ідентифікатором
Попередні умови	Отримання авторизаційного токена користувача для відправки запиту до сервісу накладання електронного підпису.
Вихідні умови	Отриманий файл електронного підпису до електронного документа

Таблиця 3.7

Варіант використання «Отримати список електронних документів з цифровим підписом»

Найменування	Отримати список електронних документів з цифровим підписом
Первинний актор	Користувач
Інші актори	Відсутні

Продовження табл. 3.7

Опис	Користувач має можливість відправити запит на отримання списку електронних документів
Попередні умови	Відкрита сторінка зі списком електронних документів та бути авторизованим у системі
Вихідні умови	Отриманий список електронних документів з цифровим підписом

Таблиця 3.8

Варіант використання «Завантажити електронний документ»

Найменування	Завантажити електронний документ
Первинний актор	Користувач
Інші актори	Відсутні
Опис	Користувач має можливість завантажити електронний документ
Попередні умови	Відкрита сторінка зі списком електронних документів та бути авторизованим у системі
Вихідні умови	Завантажений електронний документ

3.2.Створювання логічної структури програмного забезпечення

3.2.1. Створювання статичної логічної структури програмного засобу

Розробка програмного забезпечення для збереження електронних документів з цифровим підписом ставить складу задачу для реалізації та вибору правильно архітектури програмного забезпечення [27]. Сучасні темпи розвиту створюють нові підходи, які іноді не випробовують себе на практиці, але так як мова йде про важливі електроні документи системи повинна бути адаптована до написання модульних тестів та інтеграційних,

щоб визначити можливі помилки ще на етапі розробки. В результаті обставин було промодельовано майбутню архітектуру за допомогою діаграми класів у нотації UML. Даний засіб наочно зображує обсяг роботи і логічну структуру програмного забезпечення.

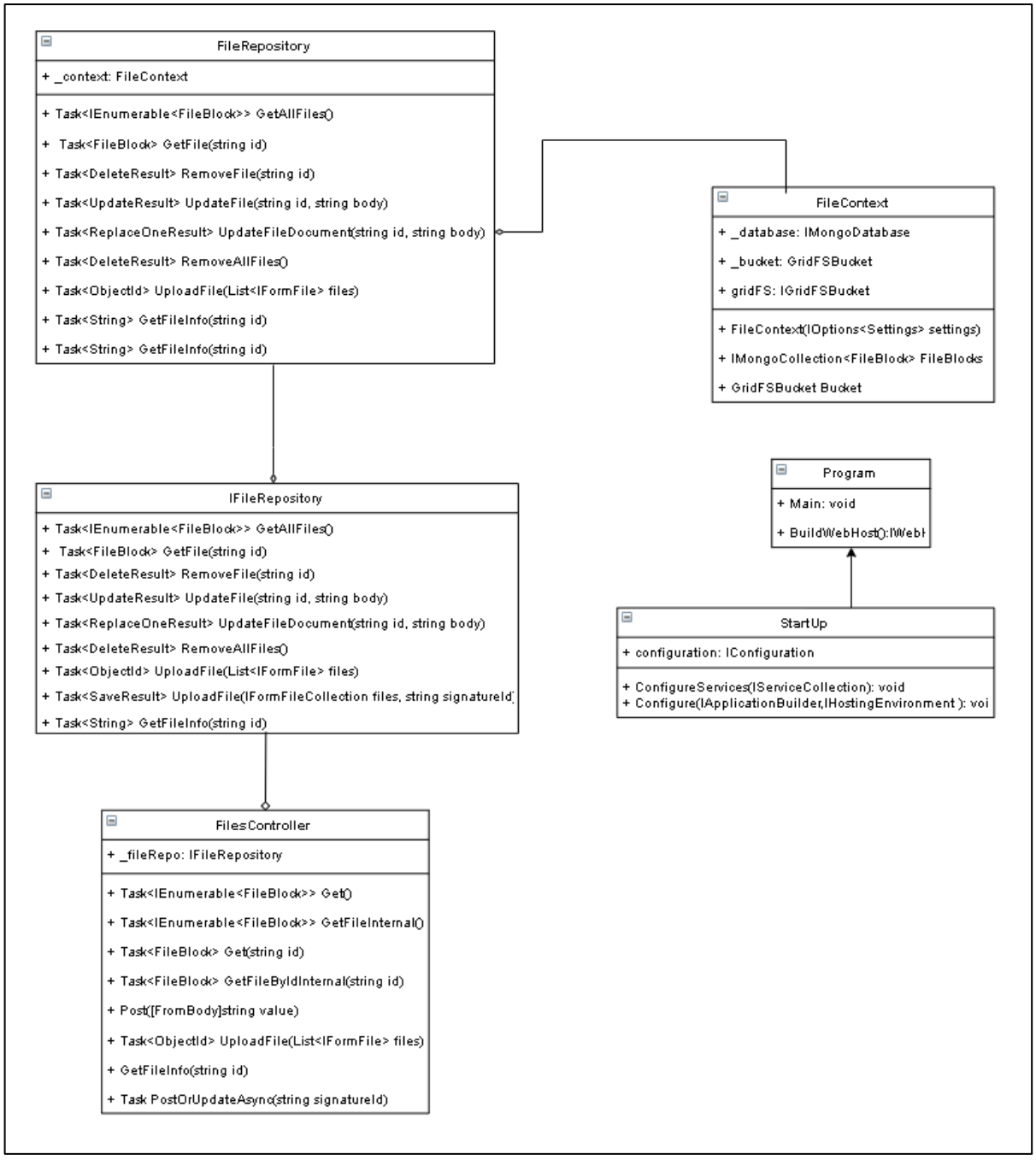


Рис. 3.2. Статична логічна структура програмного забезпечення

Класи, що використовуються в програмному забезпеченні для зберігання електронних документів наведено в табл. 3.9 [27].

Таблиця 3.9

Список класів програмного застосунку

Клас	Опис
StartUp	Клас для створення контейнеру сервісів, які реалізуються слабо в’язаний код.
FileRepository	Клас, який реалізую основну функціональність для роботи з електронними документами, та процеси перевірки вхідних запиті на сервер.
FileBlock	Клас для створення структури даних для блоку у блокчейні.
FileContext	Клас для доступу до бази даних та реалізації функціональності для зберігання електронних документів.
FilesController	Клас для створення інтерфейсів доступу до веб-сервера реалізуючи методи за принципом REST технології.
Settings	Клас для збереження основних налаштувань сервера
GridFSBuket	Клас, який реалізує функціональність для розбиття одного файлу на частини.

Список основних методів класів описаних в таблиці 3.9 наведено в табл. 3.10 – 3.1.

Таблиця 3.10

Список основних методів класу StartUp

Тип даних	Метод та опис
Void	Configure () Метод викликається під час виконання, використовується для налаштування контуру HTTP запиту.
Void	ConfigureServices (IServiceCollection services) Метод викликається під час виконання. Використовується, щоб додати послуги до контейнера.

Таблиця 3.11

Список основних методів класу FileRepository

Тип даних	Метод та опис
Task<IEnumerable<FileBlock>>	GetAllFiles () Отримання всіх електронних документів.
Task<FileBlock>	GetFile (string id) Отримання одного електронного документу за його ідентифікатором
воид	AddFile (FileBlock item) Додовання нового блоку до бази даних
Task<UpdateResult>	UpdateFile (string id, string body) Оновлення даних електронного документу за його ідентифікатором
Task<ObjectId>	UploadFile (List<IFormFile> files) Завантаження електронних документів

Продовження табл. 3.11

Task<SaveResult>	UploadFile (IFormFileCollection files, string signatureIdid) Завантаження електрооних документів разом з його електронним підписом
Task<String>	GetFileInfo (string id) Отримання інормаціх про електронний документ

Таблиця 3.12

Список основних методів класу FilesController

Тип даних	Інтерфейс доступу	Метод та опис
IEnumerable<FileBlock>	api/Files	Get () Метод реалізування інтерфейсу доступу до списку електронних документів.
IEnumerable<FileBlock>	api/Files/id	Get (string id) Метод реалізування інтерфейсу доступу отримання одного електронного документу
String	api/Files/uploadfile	Task<ObjectId> UploadFile(List<IFormFile > files) Метод реалізування інтрефейсу доступу для завантаження електронного документ у з цифровим підписом.

Продовження табл. 3.12

String	api/Files/getinfo	GetFileInfo(string id) Метод реалізування інтрефесу доступу для отримання інформаціх про електронний документ

3.2.2. Створювання динамічної логічної структури програмного забезпечення

Для наочного зображення та розуміння специфікації виконуваної поведінки системи для збереження електронних документів побудована діаграма діяльності на рис 3.3 [27].

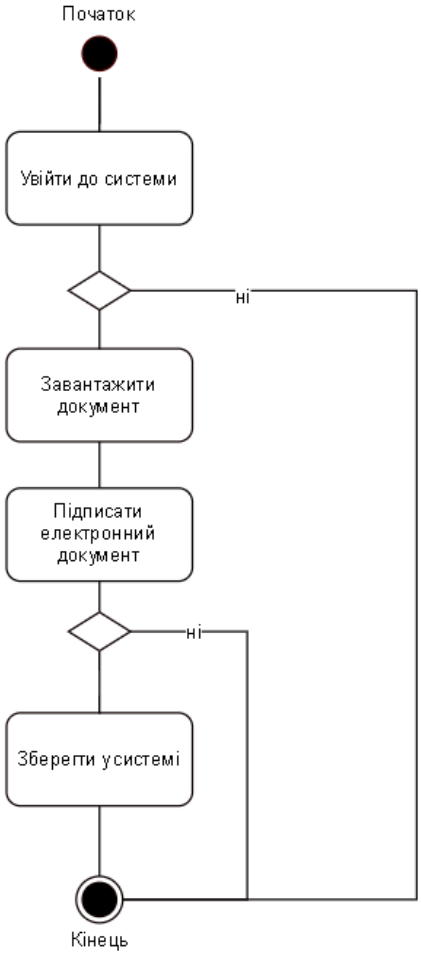


Рис. 3.3 Динамічна логічна структурна програмного забезпечення

3.3.Створювання фізичної структури програмного забезпечення

Для визначення фізичної структури програмного забезпечення побудовано діаграму компонентів, яка зображує розбиття системі на структурні одиниці та залежності між ними. В даному випадку в якості компонентів виступають файли та папки рис 3.4 [27].

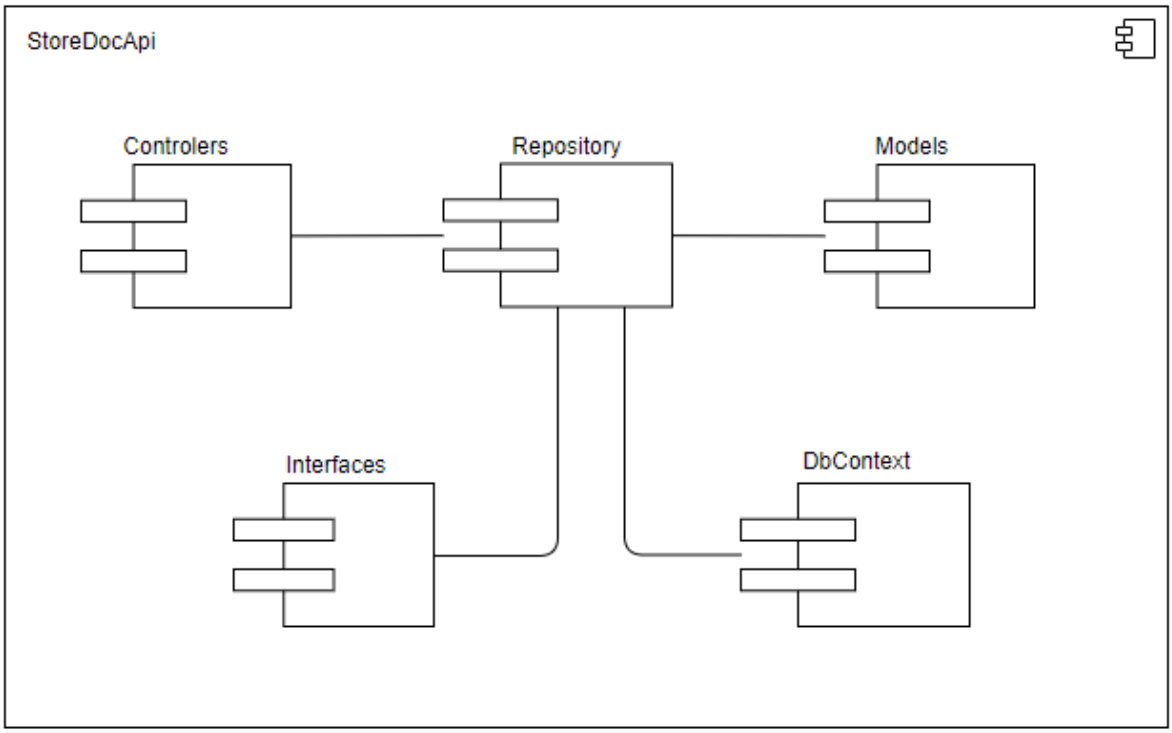


Рис. 3.4 Фізична структура програмного забезпечення

До структури Controllers поміщаються класи контролерів. Класи контролерів можуть розміщуватися де завгодно, оскільки вони компілюються в одну збірку. Класи моделей, розміщуються у Models, сюди поміщаються класи, які описують об’єкти бізнес-логіки програмного забезпечення. Класи моделей можуть бути визначені де завгодно в поточному проекті або взагалі винесені в окремий проект. Класи для роботи з базою даних поміщені до DbContext.

Для реалізації рішення інтеграції, що спрощує розширення можливостей системи, при якому потік управління програми

контролюється фреймворком, використаний шаблон проектування Repository, який розділяється на два компоненти. Данна структура містить класи для роботи з основним функціоналом програмного забезпечення та реалізування бізнес логіки [27].

Для створення фізичних структури проекту, який базується на використанні фреймворку Angular, було використано класичну організацію файлів [33]. Модулі сторінок обладнані маршрутизацією і призначені для того, щоб логічно розділити області вашої програми. Модулі сторінок завантажуються один раз в головному модулі, на малюнку зображено як app.component [34].

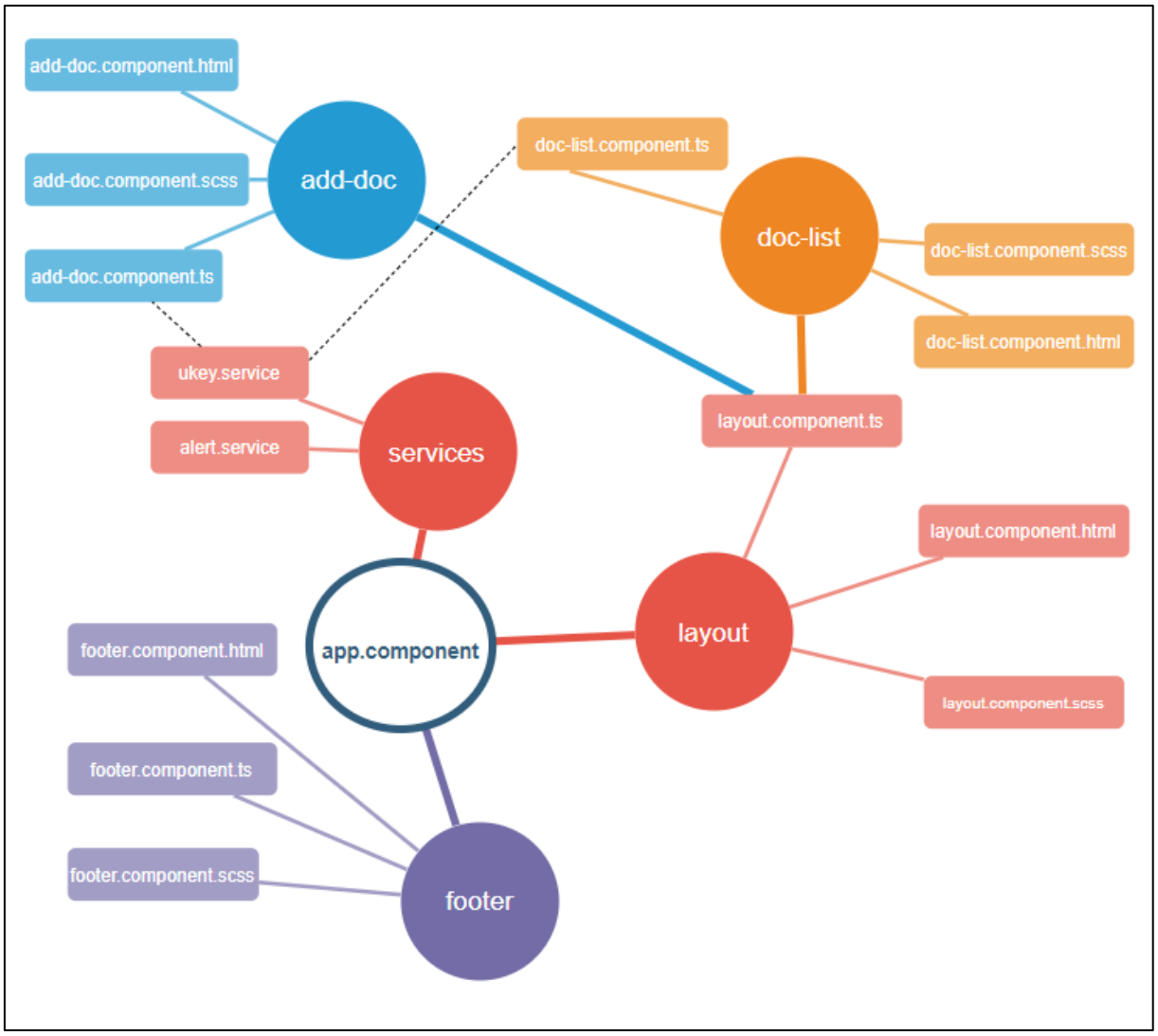


Рис. 3.4 Структурна схема компонентів

На рисунку 3.4 видно, що кожний модуль має принаймні один компонент - кореневий компонент, який з'єднує ієрархію компонент з об'єктною моделлю документа сторінки (DOM). Кожен компонент визначає клас, який містить дані програми та логіку, і пов'язаний з шаблоном HTML, який визначає представлення, яке відображатиметься у цільовому середовищі [35]. Для розуміння, в якому файлі, що знаходиться було збережено їх розширення.

3.4. Висновки

Результатами специфікування вимог до програмного забезпечення для збереження електронних документів з цифровим підписом є визначена взаємодія користувача та програмного забезпечення використовуючи діаграму діяльності за нотацією UML. Було визначено основних акторів системи та зв’язки між ними та компонентами програмного забезпечення.

Визначна статична та динамічна логічна структура використовуючи діаграму класів. Результатами є опис програмних компонентів та взаємодія між ними.

Фізична структура представлена діаграмою компонентів. Визначена файлова структура відносно задач, які повинна вирішувати програмне забезпечення для збереження електронних документів.

Основні результати розділу опубліковано в [27].

4 РОБОЧИЙ ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ЕЛЕКТРОННИХ ДОКУМЕНТІВ З ЦИФРОВИМ ПІДПИСОМ

4.1.Реалізування програмного забезпечення

Для розробки програмного забезпечення для збереження електронних документів з цифровим підписом було обьрано наступні технології: ASP.NET, .NET Core 2.0, Angular 7, MongoDB, Node js.

Платформа ASP.NET являє собою технологію, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів. ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом ASP.NET Core надає такі переваги [36]:

- Єдиний підхід для створення веб-інтерфейсу та WebApi.
- Архітектура адаптивна для проведення модульного тестування.
- Можливість розробки та запуску в Windows, MacOS та Linux.
- Вбудована ін'єкція залежностей.
- Легкий, високопродуктивний та модульний HTTP-запит.
- Можливість хостингу в IIS, Nginx, Apache, Docker або самостійному хості у вашому власному процесі.
- Інструмент, що спрощує сучасну веб-розробку [37].

Angular представляє фреймворк від компанії Google для створення клієнтських додатків. Перш за все він націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків. В цьому плані Angular є спадкоємцем іншого фреймворка AngularJS. Однією з ключових особливостей Angular є те, що він використовує в якості мови програмування TypeScript. Данная мова програмування надає великі

переваги на класичним використанням JavaScript, обумовлено це тим, що в ньому реалізована строга типізація типів даних, як наслідок виникнення помилки в коді зменшується [38]. Angular надає такі переваги:

- модульність, тобто організацією вашого коду в незалежних частинах, так для повного забезпечення повторного використання компонентів. Це робить тестування, оновлення та обслуговування додатка набагато зручнішим;
- вбудований механізм анімацій, покращує ефективність та заощаджує час на розробку;
- інтерфейс командного рядка полегшує процес розробки додатка, завдяки використанню коротких та зрозумілих команд для створення потрібних керуючих компонентів [39].

MongoDB - документоорієнтована система управління базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері [40]. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність. MongoDb надає такі переваги:

- можливість зберігати великі дані, поширюючи їх на кілька серверів, підключених до програми. Якщо сервер не може впоратися з такими великими даними, то не буде ніякого стану помилки;
- індексування доступ до документів. Отже, він забезпечує швидку відповідь запиту. Швидкість MongoDB в 100 разів швидше, ніж реляційна база даних;
- Реплікація та gridFS. Ці функції допомагають збільшити доступність даних у MongoDB. Отже, продуктивність висока;

– зберігання будь-якого типу даних у окремому документі. Ця річ дає нам гнучкість і свободу зберігання даних різних типів [41].

Вибір представлених технологій також був обумовлений тим, що вони всі є безкоштовні та не потребують витрат для їх використання, а спільнота розробників достатньо велика, щоб вирішити можливі проблеми, зв'язаних в внутрішньою реалізацією використовуваних бібліотек.

Для розробки погромного забезпечення використовуючи приведені вище технології використовувалися Visual Studio та Visual Studio Code, для написання коду. Вибір даних середовищ розробки був обумовлений тим, що вони не повністю адаптованими під обрані технології, оскільки розроблювалися однією компанією Microsoft. Для перегляду та маніпулюванням даними у базі даних було обрано Robo 3T, даний десктопний застосунок надає мінімальний функціонал, але його достатньо для поставлених задач.

Реалізація функціоналу програмного забезпечення був спланований таким чином, що усі необхідні модулі реалізовувалися поступово та одночасно. Для наочного зображення виконання поставлених задач обрано Trello, частину задач зображено на рисунку [42].

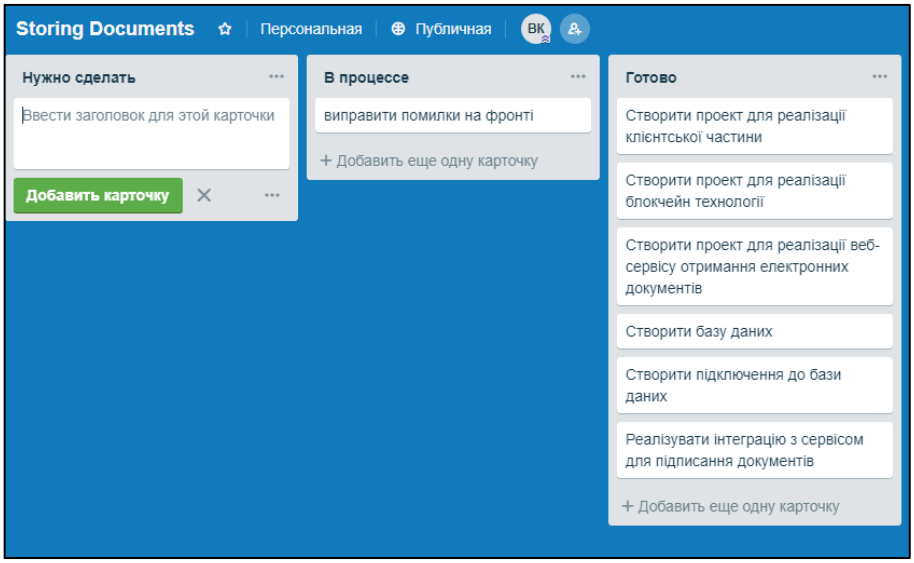


Рис. 4.1. Поставлені задачі для реалізації програмного забезпечення для збереження електронних документів

Функціональні вимоги задають поведінку розроблюваного програмного забезпечення, тому доцільно провести аналіз, щоб запобігти внутрішніх конфліктів системи [43].

Вимоги до системи наведено нижче.

Функціональні вимоги:

- завантаження електронного документу;
- підписання електронного документу;
- збереження електронного документу;
- можливість перегляду збережених електронних документів на веб-інтерфейсі.

Говорячи про нефункціональних вимогах, найчастіше говорять про атрибути якості (тобто вимогах, що визначають якісні характеристики розроблюваного програмного забезпечення або системи, такі як продуктивність, надійність, масштабованість) [44]. До веб-застосунку для збереження електронних документів висуваються наступні нефункціональні вимоги:

- відображення статусу операції, за допомогою кольорового індикатора;
- обмеження для форматів електронних документів;
- відповідність веб-інтерфейсу стандартам Web 2.0;
- застосунок повинен коректно відображатися у трьох останніх версіях браузерів Google Chrome, Mozilla Firefox, у версії Internet Explorer 9+ та в Safari 6+;
- простота і зрозумілість використання;
- швидкість роботи: застосунок повинен забезпечувати відображення основної інформації в межах 5 секунд.

4.2. Тестування програмного забезпечення

4.2.1. Опис процесів тестування

Під час тестування програмного забезпечення для збереження електронного документу перевірити функціональну відповідність наступних функцій:

- завантаження електронного документую;
- підписання електронного документу;
- збереження електронного документу у системі;
- перегляд списку збережених документів;
- авторизація у системі.

Тестування програмного виконується у ручному режимі, не використовуючи ніяких засобів автоматизації. Ручне тестування - самий низькорівневий та простий тип тестування, що не вимагають великої кількості додаткових знань. Тим не менш, перед тим, як автоматизувати тестування будь-якого додатка, необхідно спочатку виконати серію тестів вручну [45]. Для проведення даного виду тестування булим сформульовані класні сценарії використання програмного забезпечення, описані у розділі 3. Дані сценарії це артефакти, що описують сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини [46].

4.2.2. Опис контрольного прикладу

При тестуванні програмного забезпечення було перевірено функціональність розроблюваного програмного засобу відповідно до вимог у підрозділі 1.3. В табл. 4.1 - 4.3 наведено інформацію про тестування основних варіантів використання та їх результати зображені на рис. 4.2 та 4.3.

Таблиця 4.1

Перевірка завантаження електронного документу

Мета тесту	Перевірка можливості завантажити електронний документ
Початковий стан	Користувач авторизований у системі та заходиться на сторінці завантаження та збереження електронного документу.
Вхідні дані	Електронний документ розширення *.docx або *.pdf
Схема проведення тесту	Перейти на веб-застосунок та перетягнути бажаний для завантаження файл
Очікуваний результат	Повернення відповіді від веб-застосунку про успішне завантаження електронного документу
Стан програмного продукту після проведення випробувань	Електронний документ завантажений

Після перевірки контрольного прикладу «Перевірка підписання електронного документу» при роботі з програмним забезпеченням було отримано результат, що можна побачити на рис. 4.2.

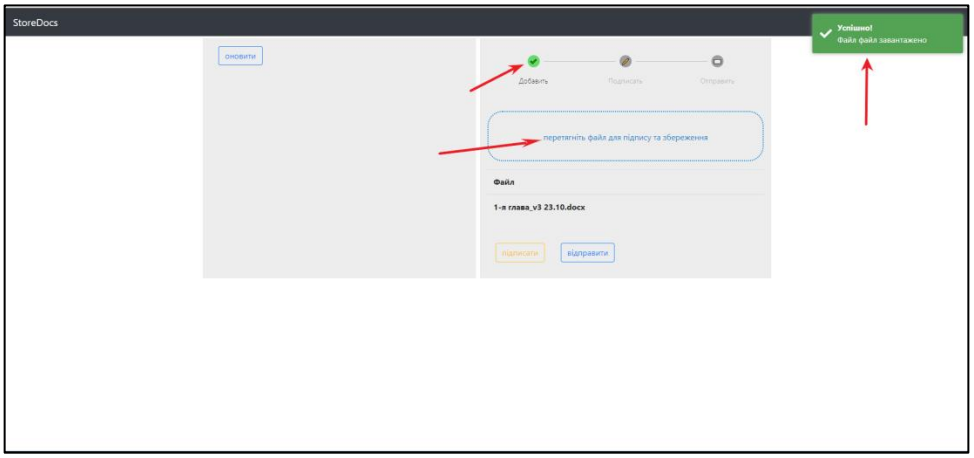


Рис. 4.2. Результат завантаження електронного документу

Таблиця 4.2

Перевірка підписання електронного документу

Мета тесту	Перевірка можливості підписати електронний документ за допомогою сервісу накладання електронного підпису
Початковий стан	Користувач авторизований у системі та звантажив електронний документ за допомогою функціоналу перетягування документу у веб-браузер
Вхідні дані	Завантажений електронний документ розширення
Схема проведення тесту	Перейти на веб-застосунок та натиснути на кнопку «підписати» на веб інтерфейс, на панелі підписання та завантаження електронного документу
Очікуваний результат	Повернення відповіді від сервісам підписання документів «Електронний документ підписаний» та відповідна індикація на верхній панелі зеленим кольором, що свідчить про успішність операції
Стан програмного продукту після проведення випробувань	Електронний документ підписаний

Після перевірки контрольного прикладу «Перевірка підписання електронного документу» при роботі з програмним забезпеченням було отримано результат, що можна побачити на рис. 4.3.

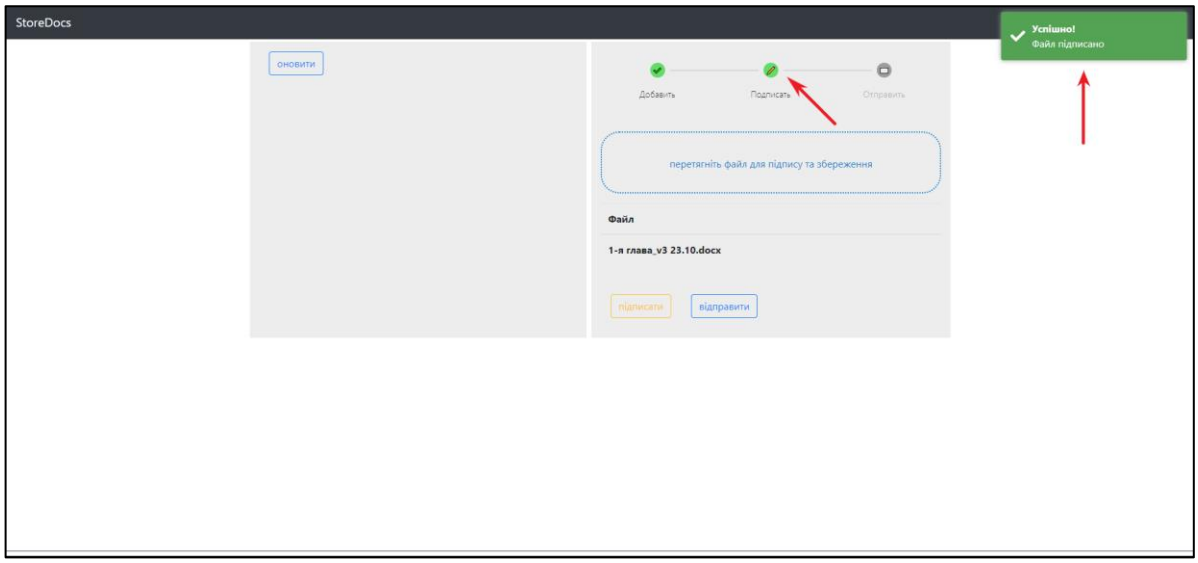


Рис. 4.3. Результат підписання електронного документу

Таблиця 4.3

Перевірка завантаження електронного документу

Мета тесту	Перевірка можливості звантажити електронний документ
Початковий стан	Користувач авторизований у системі, звантажив та підписав електронний документ
Вхідні дані	Підписаний електронний документ
Схема проведення тесту	Перейти на веб-застосунок та натиснути на кнопку «завантажити» на веб інтерфейс, на панелі підписання та завантаження електронного документу
Очікуваний результат	Повернення відповіді від сервера «Успішно завантажено» та відповідна індикація на верхній панелі зеленим кольором, що свідчить про успішність операції
Стан програмного продукту після проведення випробувань	Електронний документ завантажений

Після перевірки контрольного прикладу «Перевірка завантаження електронного документу» при роботі з програмним забезпеченням було отримано результат, що можна побачити на рис. 4.4.

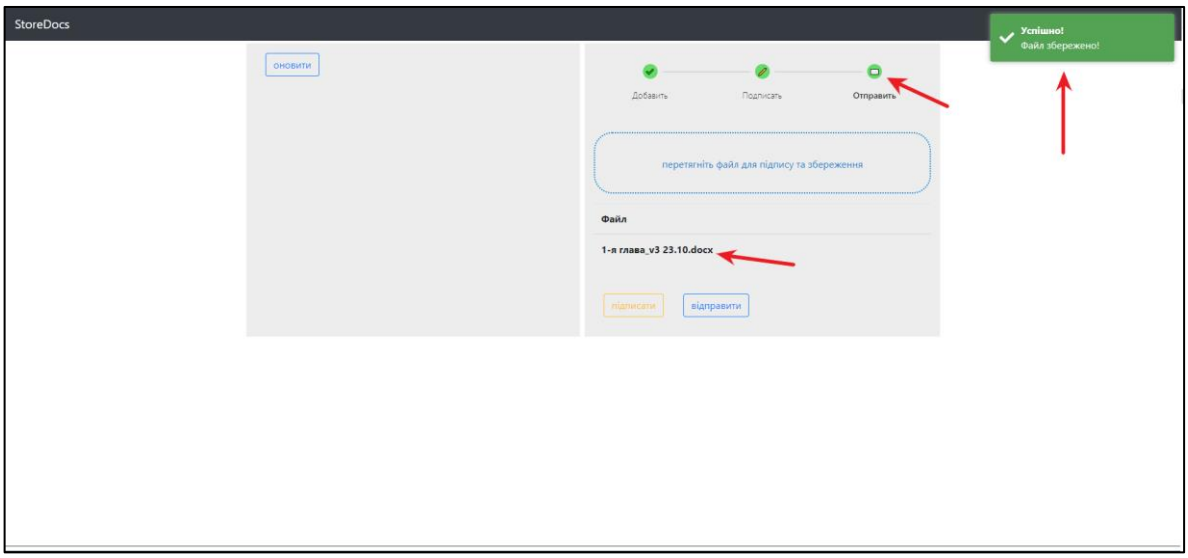


Рис. 4.4. Результат збереження електронного документу

4.3.Використання програмного забезпечення

Для користувача була розроблена клієнтська частин, яка є веб-інтерфейсом, для її роботи достатньо встановленого веб-браузера. Для нормальної роботи програмного забезпечення достатньо встановити один з наступних браузерів: Google Chrome, Mozilla Firefox, Internet Explorer 9+ та в Safari 6+. Оскільки модель збереження електронних документів схожа з хмарним сховище, ніяких додаткових налаштувань від користувача не потрібно.

Для розгортання програного забезпечення необхідно наплутувати середовище, яке здатно виконувати команди закладені у веб сервіс. Традиційно додатки ASP.NET розгорталися на веб-сервері IIS. Однак оскільки ASP.NET Core має кроссплатформенную природу, треба було відв'язати ASP.NET Core від IIS і в цілому від Windows. І на даний момент ASP.NET Core підтримує розгортання програми на стандартних веб-

серверах IIS і IIS Express, а також надає можливість запускати додаток без IIS в рамках власного процесу за допомогою двох додаткових http-серверів, які йдуть разом з ASP.NET Core: Microsoft.AspNetCore.Server.HttpSys, Kestrel [47].

До складу файлів необхідних для старту веб-сервіса входять (див. рис. 4.5):

– StoreDocApi.deps.json, необхідний для завантаження необхідних бібліотек та залежностей, які завантажувались спрощення реалізації функціоналу.
– StoreDocApi.dll, містить основний код, який здатний обробляти вхідні запити та давати відповідь
– StoreDocApi.runtimeconfig.dev.json, який визначає загальну тривалість роботи, яку очікує програма, а також інші параметри конфігурації для середовища виконання (наприклад, тип збирання сміття), це один із результатів публікації програми, який готовий до розгортання на сервері [48].

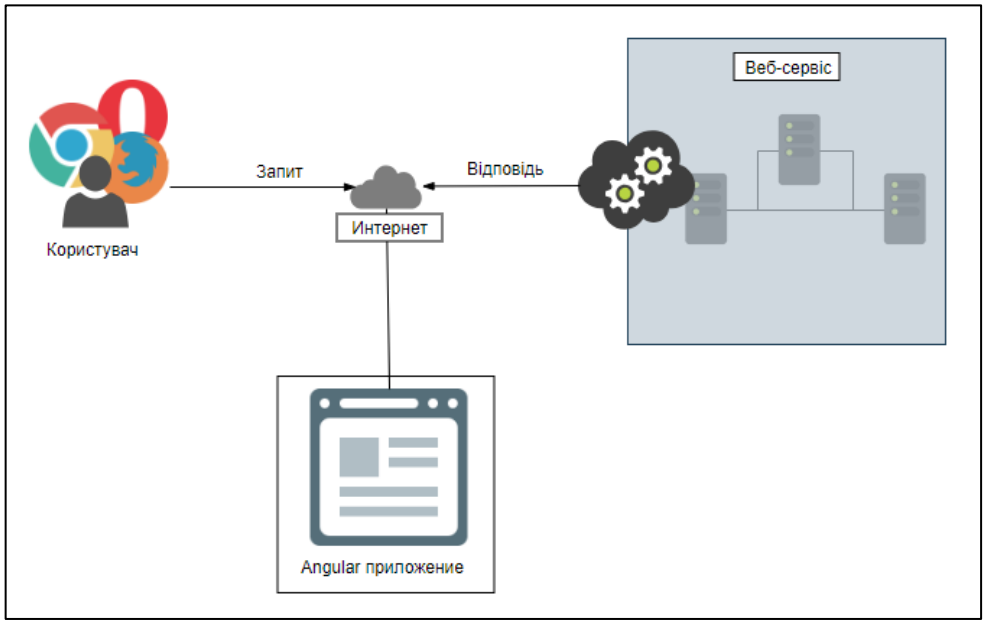


Рис. 4.5. Схематичне зображення результату розгортання програмного забезпечення на серверах

4.4.Висновки

У даному розділі було наведено опис програмної реалізації програмного забезпечення для зберігання електронних документів за технологіє блокчейн [27]. Дана програмна реалізація створена для застосування в сфері, де має місце критично важлива безпека збереження документів з накладеним цифровим підписом. Програмне забезпечення було розроблене з використанням технологій ASP.NET Core, Angular 6, MongoDb, відповідно до методики, наведеної в третьому розділі.

Тестування було проведено із використанням класного модульного тестування програмного забезпечення. Були реалізовані інтеграційні тести, які перевіряють на коректність роботи всієї системи, так як зв'язок між модулями критично важливий.

Наведено приклади використання програмного забезпечення із контрольними прикладами та їх результатами, за попередньо сформованим тест-планом.

Основні результати розділу опубліковано в [27].

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ЕЛЕКТРОННИХ ДОКУМЕНТІВ З ЦИФРОВИМ ПІДПИСОМ

5.1.Опис ідеї проекту

На сьогоднішній день електронні документи замінюють паперові майже у всіх сферах, де документ має якусь юридичну силу або доказ проведеної роботи такі як доставка, тобто логістичні компанії, безпосередньо юридичні компанії, закупка медпрепаратів, введення історій хвороби, згоди на відповідні процедури, дані документи зустрічаються у медицині. Тому можна зробити висновок, що автоматизація та електризація нашого повсякденного життя торкнулося майже всіх сфер діяльності.

Усі вище приведені допустимі документі потрібно зберігати у сховищах, де вони б надійно зберігалися, а власники приватної інформації зазначених в них могли бути спокійними.

Паперові документи в більшості випадках зберігаються у архівах, що в свою чергу є не досить безпечно. Тому використання електронних архівів та систем документообігу почали вирішувати проблеми паперових сховищ. Одними з головних проблем є:

- виділене місце для зберігання, що в тягне за собою питання обслуговування;
- безпека, звичайний пожар може знищити інформацію;
- швидкий доступ, пошук необхідного документа може зайняти багато часу;
- людський фактор.

Говорячи про електроні сховища, такими сховищами можуть виступати закриті сервери, які не мають доступу до інтернету, тобто в закритих мережах, різні криптографічно-захищені носії доступ до яких здійснюються за допомогою приватного ключа та бази даних. При таких

видах зберігання не має повної впевненості, що данні не будуть змінені, або видалені, адже фактор людської помилки теж присутній і його потрібно враховувати.

Обслуговування баз даних зазвичай займаються адміністратори, вони можуть змінювати, переглядати та загалом робити будь-які спекуляції з даними, які в ній зберігаються. Будь-яка зміна інформації або командний запит, не логується у файл змін або таблицю, окрім використання збережених процедур, де зазвичай розробники передбачають механізм збереження історії змін електронних документів. Наявні проблеми зручно зобразити за допомогою дерева-проблем [49].

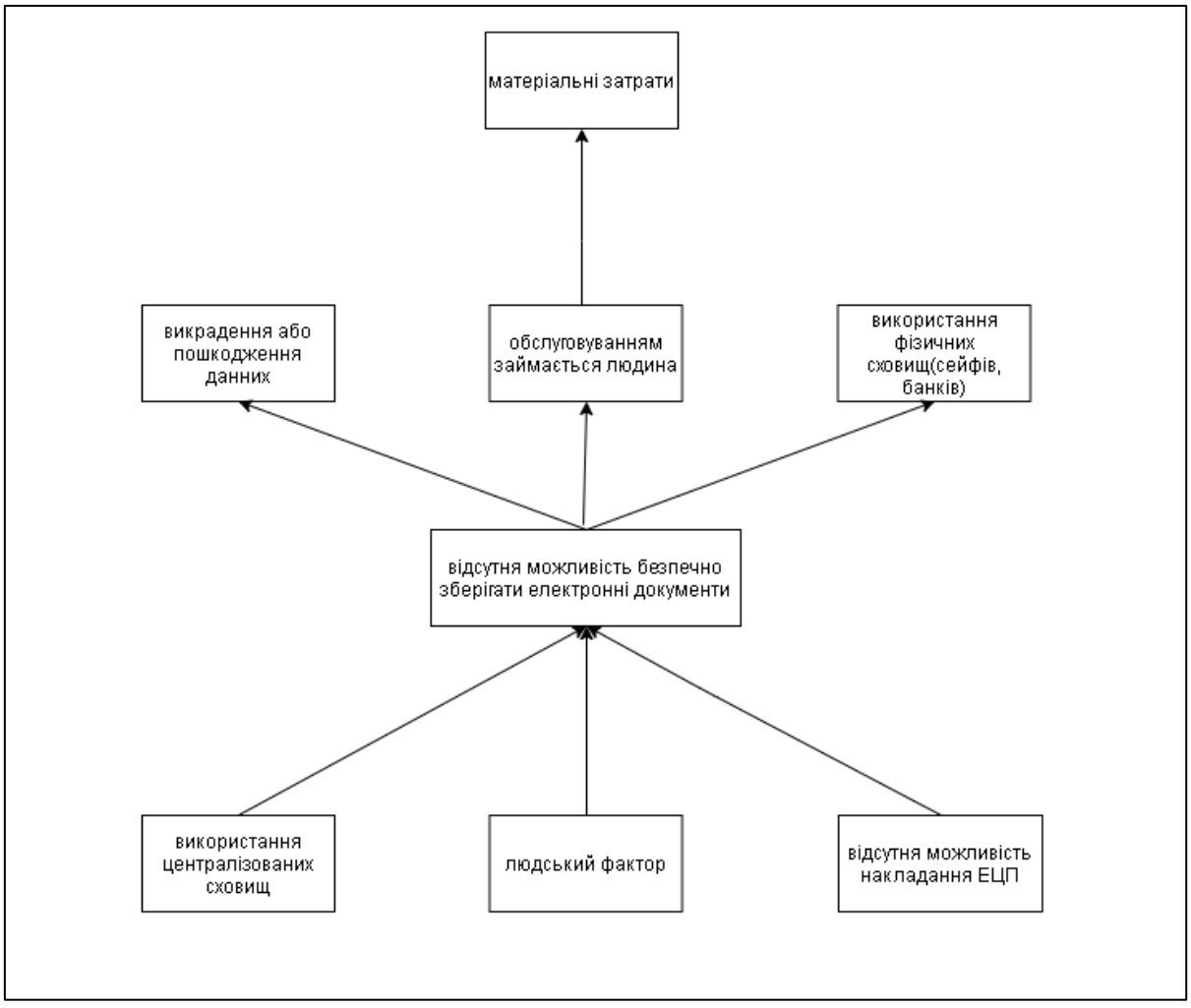


Рис. 5.1 - Дерево проблем

Саме тому пропонується використовувати для збереження документів технологія, яка успішно використовується у фінансових установах та стартапах – блокчейн.

За допомогою реалізації децентралізованого сховища, проблема з переносом даних або копіювання буде проходити набагато безпечніше та менш ресурсозатратним. При виникненні можливості переходу на більш продуктивні обчислювальні машини, процес переходу не буде потребувати вимкнення системи та призупиняти роботу користувачів, навіть у нічний період.

5.2.Технологічний аудит ідеї проекту

Результатом науково-дослідницької роботи є програмний засіб для зберігання договорів з цифровим підписом за технологією блокчейн, представлене у вигляді веб-сервісу [27].

Даний сервіс виконує роль порталу, де користувач може підписувати один документ за допомогою електронних ключів. Також сервіс надає можливість зберігати документи створені та підписані однією людиною, на увазі мається прикази чи будь-який документ, який має юридичну силу.

Основним плюсом використання даної технології є те, що електронно створенні документі не можливо буде знищити, а дані про авторство створення та підписання документа не можливо буде підробити. Даний сервіс не може бути централізованим, тому немає різниці де будуть розміщувати сервера системи. Лише одним обмеженням є обсяг даних, який повинен зберігати всю історію потрапляння документів від користувачів.

Слід зазначити, що відклик від серверної частини сервісу є дещо довшим ніж прийнято рахувати у подібних системах. Система повинна бути абсолютно синхронізована, щоб дати остаточну відповідь користувачу, що його данні успішно збережені. Тому слід враховувати,

що користувачі будуть готові, що процес створення документа може зайняти час.

Юридичні та фізичні особи зможуть підписувати договори між собою без приватних зустрічей, а у справжності документа не може бути сумніву так як має бути накладений електронний підпис, власником сертифікату, якого відповідно є користувач [50].

При внесенні документу до сховища вносяться данні і про сертифікат, зокрема ким був виданий, кому, якого числа та до якого числа дійсний. В залежності від країни, термін дії може бути різним, але в більшості випадках він триває один календарний рік. В наслідок не довготривалої дії сертифікату є доречним ці дані зберігати. В майбутньому не буде складності довести чи навести довідку про те, що дії чи операції, які були підтверджені електронним договором є справжніми [51].

Кластер підібраних технологій відповідає поточним тенденціям, шо в свою чергу буде спрямовувати систему на вдосконалення, а процес підбору персоналу для підтримки буде менш затратним.

Так як в основу реалізації електронного архіву документів було покладено технологію блокчейн, основною особливістю, якої є те що данні одночасно зберігаються на декількох серверах одночасно. Використання даного принципу збереження даних накладає ряд обмежень, зі сторони використовуваної пам’яті, але в майбутньому даний ресурс будем все менше впливати на роботу будь-яких додатків.

Серверна частина реалізована за принципом RESTfull api, що дає змогу в залежності від подальших обставин зробити API відкритим [52]. Клієнтська частина є незалежною від серверної і принципової технології для реалізації нема. Робота користувача у системі можна розбити на модулі:

- власний кабінет;
- робота з документами;

– цифровий підпис.

При ознайомлені з кількістю наявних модулів, можна зробити висновок, що інтерфейс буде не навантаженими та дуже простим.

В ході роботи сервісу, збирається аналітика, яка згодом дасть достовірну інформацію про дії користувача та функції, якими він користується в цілому. Це дасть змогу покращити сервіс більш доцільно, якщо в цьому буде сенс.

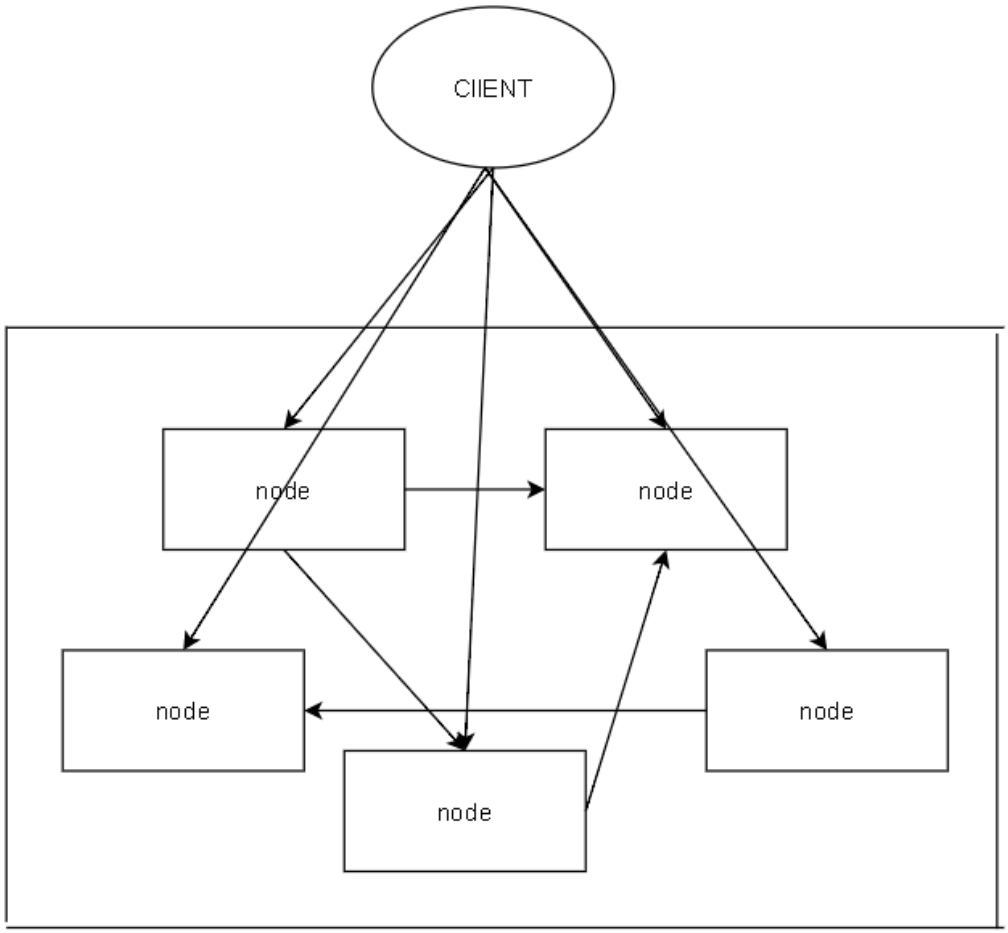


Рис.5.2. Схема зв’язку клієнтської та серверної частини

На малюнку зображено спрощену схему взаємодії двох головних частин сервісу:

– клієнтська частина, буде реалізована як одно сторінковий додаток, із застосуванням Angular 5;

– серверна частина, в ній реалізовано головну цінність продукту, Блокчейн, на зображенні видно, що система складаються із декількох складових, що абсолютно подібні між собою, які і створюють децентралізованість.

За допомогою реалізації серверної частини таким чином, даний принцип забезпечить високий рівень захищеності донних від змін, та буде створювати велику перевагу у порівняні із вже існуючими аналогами. На сьогоднішній день не існує готових рішень, які використовувалися на продуктовому середовищі децентралізовані сховища для збереження електронних документів.

Підсумовуючи опис характеристики рішення можна стверджувати, що спроектоване рішення може застосовуватися людьми, які мають справу із електронними документами із використанням електронного підпису для передачі іншим особам. Рішення не потребує додаткових вмінь та навичок для використання та встановлення додаткових сервісів.

5.3.Аналіз ринкових можливостей запуску стартап-проекту

Представлений принцип, використання технології блокчейн у збереженні електронних документів, має суттєву перевагу у безпеці збереженні даних. Насамперед на увазі мається, те що неможливо буде виконати маніпуляції з даними без відома всієї системи. Тобто якщо дані збереженого документа у системі захоче хтось змінити, то данні прийдеться змінити одночасно на всіх серверах. Для користувача класична поведінка у системі документообігу не зміниться, але серверна частина суттєво різниця з класичним розумінням клієнт серверної архітектури.

При обиранні засобів зберігання своїх електронних документів клієнтам важливо декілька питання:

– Де зберігати створенні електронні документи?

– У якому форматі зберігати? Це важливо так як процес розвитку файлобмінних сервісів швидко розвіваються, а це напряму впливає на можливість адекватно відобразити інформацію, прочитати, отримати данні про електронний підпис закладений у нього. Тому важко казати, як сьогоднішні формати відтворити завтра.
– Як забезпечити тривале збереження документа? Якщо твердотілі накопичувачі, мають дуже обмежену кількість записів, а жорстко тілі дуже уступають по швидкості зчитування та запису.

Процес зберігання відбувається за допомогою не реляційних баз даних або і власно створеного формату, де данні будуть організовані таким чином, що пошук бажаного документу відбувався швидше.

Форматом збереження виступає широковживаний JSON, він легко інтегрується з нереляційними базами даних, дані в ньому зберігаються структуровано і на сьогоднішній день він майже повністю витіснив XML.

Кожен створений документ вноситься у базу із відповідним заголовком, який буде вказувати на тип документу (договір про оренду, покупку тд.). Всі необхідні реквізити документу будуть зберігаються, а тому проблем із зчитування специфічних документів не буде.

При виходженні з ладу одного з вузлів всієї системи, можна буде змінити її на більш новий носій, а система при виявленні нового учасника однорангової сітки створе копію всього накопиченого матеріалу та відновить інформацію.

Для створення рішення, головною функцією, є зберігання та захищення електронних документів мають бути залучені фахівці різних галузей інформаційних технологій. В першу чергу для старту проекту потрібно виконати робото з аналізу існуючих рішень, якщо такі існують, та оцінити ключові моменти.

Для розробки, тестування та підтримки продукту сформована команда, в якій повинно бути налагоджений процес релізів та видачі нових

версій для альфа та бета тестувань, що може позитивно відобразитися на робот продукту в майбутньому.

Для контролювання версій використовується популярна система Git, з приватним репозиторієм, так як відкритий код в даній галузі є не доречним, з точки зору безпеки, сума за рік буде становити 100$.

Розроблений сервіс включає в себе функціонал, який схожий з конкурентоспроможними компаніями, тому слід виділити цінність за яку клієнти готові віддавати гроші:

- стабільна робота, цілодобовий доступ до власних файлів;
- безпека даних;
- сервіс сповіщення про перегляд власно створеного документу, чи отримання документу від сторонньої особи для підписання, так як деякі договори мають часові обмеження.

Процес продажу буде відбуватися безпосередньо через взаємодію з користувачам даного сервісу. При реєстрації себе як фізичної особи чи себе, як юридичної буде запропонований план оплати. В даному випадку користувач повинен сплатити за кількість документів в яких він виступає першим чи другим лицем. А сервіс забезпечує надійність зберігання документу у децентралізованій системі і при згоді користувача будь-який документ може бути доступний за посиланням будь-якому користувачу.

Також слід зазначити, що користувачу буде запропонований тип користування, для компанії таким чином формуються потоки прибутку. Для даного типу сервісу будуть запропоновані:

- абонентська плата (підписка), якщо користувачу необхідно підписувати та зберігати на віддаленому сервісі багато документів, то звісно йому буде вигідно обрати даний тип, який гарантує довготривалий термін, без необхідності сплачувати часто [53];

– плата за користування, якщо немає необхідності у збереженні великої кількості документів, то наприклад разове використання буде здійснюватися за даним типом оплати.

Сервіс буде спрямований на утримання клієнтів та розширення кельтської бази, тому варіант з довгостроковою підпискою буде більш заохочуваним, що дасть можливість більш вужче направляти рекламні банери та оповіщення.

Після місячного використання сервісу буде проведений аналіз за метриками, які б цікавили рекламні компанії і після цього буде сформована монетизація, що створе відповідний пасивний канал прибутку.

Розрахований дохід по місяцям протягом першого року наведено втабл. 5.1. Всі суми наведено в доларах США.

Таблиця 5.1

Доходи

	Довготривалі підписки	Використання одиничних документів	Індивідуальне налаштування	Всього
1	50	10	50	110
2	100	20	150	270
3	150	30	200	380
4	200	40	500	740
5	250	50	500	800
6	300	60	500	860
7	350	70	500	920
8	400	80	500	980
9	450	90	700	1210
10	500	100	700	1300
11	550	110	700	1330
12	600	120	800	1520
Сумарно за рік:				10420

Для реалізації децентралізованості зберігання документів, сервіс повинен зберігати документи більш ніж на трьох серверах, чим більше серверів тим можливість впливу на данні зменшується в рази, хоча обслуговування великої кількість серверів не має сенсу, так як швидкість обробки запиту на обробку документа буде збільшуватися, що в свою чергу відобразиться на зворотному зв’язку з клієнтами.

До витрат відноситься оренда та в майбутньому купівля власних серверів для розміщення програмного забезпечення (див. табл. 5.2). Керувати та слідкувати за стабільною роботою буде системний адміністратор на постійній основі.

Для підтримки високої якості продукту буде проводитися для кожної мажорної чи мінорної версії тестування:

- мануальне, на даному етапі буде протестований інтерфейс користувача та коректність роботи валідації існуючих форм;
- функціональне – чи відповідають ліх зроблені користувачами реальним очікуванням;
- тестування безпеки- даний аспект є критичним так, як помилка на даного виду буде коштувати дуже дорого для її вирішення;
- тестування продуктивності.

Коли продукт буде доведено до стану який можна буде висувати у пілотне користування буде підбиратися команда для підтримки продукту.

Після впровадження продукту на продуктову середовища з реальними користувачами буде сформована група підтримки, яка буде займатися покращенням існуючого функціоналу та відтворенням та рішенням проблем які виникають у користувачів. Після формування такої команди буде виділятися відсоток для заробітних плат .

Таблиця 5.2

Витрати

	Технічна підтримка	Оплата праці	Комунальні та адміністративні	Реклама	Всього
1	400	1400	400	1000	3500
2	40	1400	400	400	2500
3	40	1400	400	400	2500
4	40	1400	400	300	2300
5	40	1400	400	300	2300
6	40	1400	400	300	2300
7	40	1400	400	300	23500
8	40	1000	400	300	1800
9	40	1000	400	300	1800
10	40	1000	400	300	1800
11	40	1000	400	300	1800
12	40	1000	400	300	1800
				Сумарно за рік:	13250

По мірі надходження коштів з користування програмним забезпеченянм вже існуючими користувачами, великій відсоток прибутку буде виділятися на рекламні компанії.

В свою чергу заходами, які можуть вплинути позитивно на продажі може бути різні виставки, конференції, як правило участь в таких заходах не буває безкоштовною, так як передбачає за собою виставковий стенд та можливість представити свій продукт готовій публіці.

Отже можна зробити висновок, що попередні витрати будуть виділятися на фахівців, з яких можна зібрати команду для успішної реалізації так запуску продукту у пілот. Доходи не передбачуються найближчим часом після запуску так, як продукт має бути добре відтестований у бета версії, після чото вже з існуючою базою потенційних клієнтів можна формувати плани оплат.

5.4.Розроблення ринкової стратегії проекту

При реалізації проекту важливим фактором є оцінка складності та можливі проблеми, які можуть виникнути в майбутньому. Так як створення продукту несе різний вплив на оточення слід розглянути всі зацікавлені сторони.

Визначити перелік основних характеристик кожної з зацікавлених сторін, наскільки сторона зацікавлена в продукті, взаємовідносинами між ними. Для попереднього аналізу опишемо зацікавленні сторони за допомогою таблиці, метриками якої будуть виступати: вплив та влада [27, 54]. Були проаналізовані зовнішні фактори (див. табл. 5.3).

Таблиця 5.3

Аналіз зовнішніх факторів

Фактори	Можливості	Влада	Вплив		
			Влада	Інтерес	Підсумок
ЦСК (центри сертифікації ключів)	Видають персональні електронні ключі для підписів. Є довіреною особою	Залежність від рівня захисту та складності алгоритмів генерації сертифікатів	4	8	32
Юр. і фіз особи	Розширення клієнтської бази	Диктує попит	4	10	40
Соціум	Поширення компанії для більшої популярності і популярності компанії, прийняття сучасних технологій		1	2	2
Держава	Контроль податків Регулювання діяльності Введення обмежень	Обмеження згідно законів	4	3	12

Підраховуючи загальну оцінку можна зробити висновок, що основним фактором впливу будуть виступати самі клієнти, які зацікавлені у функціоналі сервісу та процесі збереження їх даних.

Тобто всі наявні потенційні особи, які у своїй діяльності використовують електронні документи та обслуговуються у центрах сертифікації.

Також можна зробити висновок, що при реалізації рекламних компаній широкою аудиторією можна залучитися комерційною підтримкою центрів сертифікацій, в них вже наявна база постійних клієнтів.

Потенційних клієнтів також можна шукати серед державних підприємств, там де можливе місце корупції. Створенні документи у сервісі будуть мати відкриту інформацію про сторони договорів, звісно, якщо цього буде бажати користувач.

Були проаналізовані внутрішні фактори (див. рис. 5.4).

Таблиця 5.4

Внутрішні фактори

Фактори	Можливості	Слабкість	Вплив		
			Влада	Інтерес	Сума
Топ - менеджер	Приймають головні рішення щодо діяльності компанії	Відповідальні за прийняття рішень	9	10	90
Сейл-менеджер	Налагоджують канали зв’язку та тісно співпрацюють з маркетологами	Відповідають за кількість каналів доходів та загальний прибуток	8	8	64
Менед-жер	Організація продуктивності роботи персоналу	Зобов'язані вирішувати завдання і проблеми, що виникають у користувачів	4	5	20

Продовження табл. 5.4

Розробники	Забезпечення якості наданого продукту	Стають головними технічними консультантами в компанії, регулюють технічну складову продукту	4	5	20
Бізнес-аналітики	Оцінюють фінансовий та операційний аспекти роботи компанії і визначають перспективи її розвитку	Помилки при аналізі діяльності та оцінки перспектив компанії	5	5	25
Маркетолог	Займаються просуванням продукту на ринку	Компанія рухається в тому напрямку розвитку, який обирають маркетологи	9	5	45

Основним впливаючим фактором продукту виступають ТОР-менеджери. Вони керують компанією і слідкують за виконанням усіх обов'язків своїх підлеглих, прямими підлеглими виступають сейл-менеджери, це ті люди, які продають продукт та виводять його на вже існуючий ринок. Вони створюють нові канали доходу та стежать за появою нових та існуючих ніш.

5.5 Розроблення маркетингової програми стартап-проекту

Так як клієнтура кожного бізнес-продукту модна сегментувати для покращення роботи застосунку та правильно направляти робочі сил команди, було проаналізовано сегменти за деякими критеріями.

Не планується використання продукту за межами україномовного та російськомовного населення.

Даний продукт локалізований також для англомовних потенційних користувачів, якщо з’явиться можливість та виклика кількість відгуків від користувачів про адаптацію під різномовну аудиторію цих відгуків, то звісно це відбудеться.

Що стосується доступу до даного додатку, то він не обмежений і не залежний від географічного місцезнаходження людини. Вільний доступ в інтернет та необхідне програмне забезпечення для накладання електронного підпису, буде достатньо для вільної роботи, але дані обмеження відносяться тільки для користувачів.

Програмний застосунок являться актуальним для людей, які мають справу із електронними документами, працюють бухгалтерами чи високо посадовими людьми у компаніях. Тобто кажучи про демографічну сегментацію ринку наших потенційних користувачів, то немає чітких вікових меж, щоб можна було точно сказати про вік. Тож сказати щось конкретне по рівень прибутку, роду занять, освіті, віросповідання, расі і національності не можна так, як даним продуктом будуть користуватися люди, які і до появи цього продукту констатували з документами у будь якому вигляді.

Після попереднього аналізу можна зробити висновок, що продукт не в змозі задовольнити всі потреби ринку на конкретні товари і тому вона повинна сконцентруватися на реалізації цього товару на сегментах ринку де інтерес до використання технологій документообігу у своїй роботі.

Як вже було зазначено у аналізі доходів та витрат основними планом прибутку будуть виступати дві бізнес моделі:

- оплата по факту використання;
- підписка.

Замість попередньої покупки певного періоду використання користувач буде оплачувати ту кількість документів, які будуть зберігатися у системі.

Вона не несе великого прибутку, тому що корчувач, який прийшов швидко у систему так само швидко може і піти, але гарні враження від використання можуть гарно відобразитися на маркетингу, що в свою чергу залучити нових людей до системи.

Суть другої моделі полягає в тому, що споживачі повинні вносити абонентську плату за доступ до послуги.

Зазначені види бізнес моделі які будуть накладатися на існуючий продук на початку свого розвитку будуть поступати freemium моделі.

Модель freemium дозволяє мати безкоштовний доступ до необмеженого використання базових функцій і передбачає плату тільки для клієнтів, для яких потрібна додаткова функціональність. Для накладання даного типу та приваблювання нових користувачів, у системі має бути реалізований весь функціонал [55].

5.6 Висновки

В даному розділі розглянуто питання практичного застосування системи для створення та зберігання електронних документі в за допомогою технології блокчейн [27].

Провівши аналіз ринку засобів для зберігання документів можна зазначити, що засіб для зберігання документів за технологією блокчейн не буде широко використовуваним серед людей, у яких відсутня необхідність збереження критично важливих документів. Відносно цього комерціалізація буде слабкою до більшої популярності технології блокчейн [27]. Сильний інтерес до даної технології пропорційний популярності розробленого засобу.

Впровадження продукту на ринок може відбутися за успішного проходження маркетингової компанії та інформування потенційних

користувачів про наявність розробки. Тому що технологія нова, недовіра до засобу висока. Створена бізнес модель не є досконалою і потребує допрацювань та додаткового аналізу, незважаючи на це можна сказати, що проект може існувати. Описані сегменти споживачів, які готові платити за вирішення даної проблеми, наявні унікальні ціннісні пропозиції для кожного сегменту. Запропонований програмний продукт має конкурентні переваги. Розраховані доходи та витрати.

Подальша імплементація залежить від оборотного зв’язку з користувачем, якщо за пів року позитивних відгуків буде достатньо для подальшої оплати роботи співробітників та матеріального забезпечення роботи продукту, то відповідно доцільно буде продовжувати та розвивати сервіс.

ВИСНОВКИ

У магістерській дисертації вирішено актуальне завдання, зокрема, розроблено програмне забезпечення для зберігання електронних документів з цифровим підписом за технологією блокчейн. При цьому отримано такі результати:

1. Проаналізовано програмні засоби для зберігання електронних документів, особливу увагу приділяється увага документа з цифровим підписом. Внаслідок такого аналізу показано необхідність створення відповідного програмного засобу зберігання екстрених документів з цифровим підписом за технологією блокчейн. На основі проаналізованої інформації сформовано функціональні та нефункціональні вимоги та поставлено цілі, які необхідно досягнути в процесі розробки.

2. Побудовано концептуальну модель програмного забезпечення зберігання електронних документів з цифровим підписом за технологією блокчейн, використання якої дозволяє формалізування його роботу на рівні функцій, процесів і потоків даних, а також сформувати та обґрунтувати варіанти використання програмного забезпечення.

3. Побудовано об’єктно-орієнтовану модель програмного забезпечення на основі формалізування його роботи на рівні функцій, процесів і потоків даних, використання якої дозволяє надати нову якість програмному забезпеченню при створенні або вдосконаленні, зокрема, забезпечити його функціональну придатність до зберігання електронних документів з цифровим підписом за технологією блокчейн.

4. Визначено характеристики та метрики якості програмного забезпечення і проведено функціональне тестування. Наведено контрольні приклади для проведення тестування основних варіантів використання. Визначено системні та апаратні вимоги до клієнтської та серверної частини розроблюваного програмного засобу. Було описано кроки, які необхідно виконати для успішного встановлення програмного засобу.

Також наведено інструкцію з користування програмним забезпеченням для адміністратора.

5. Визначено ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення стартап-проекту для розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Блокчейн [Електронний ресурс]. – Режим доступу: https://tinyurl.com/y8xonjuj.

2. Децентралізація [Електронний ресурс]. – Режим доступу: https://tinyurl.com/y8hftcax.

3. Хмарні сховища [Електронний ресурс]. – Режим доступу: https://tinyurl.com/jycnblw.

4. Хешування [Електронний ресурс]. – Режим доступу: https://tinyurl.com/jycnblw.

5. Вузол зв’язку [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Bootstrapping_node.

6. Позаофісне зберігання документів [Електронний ресурс]. – Режим доступу: https://goo.gl/SBQTTP.

7. Види цифрового підпису [Електронний ресурс]. – Режим доступу : https://goo.gl/ht5doQ.

8. Види цифрового підпису [Електронний ресурс]. – Режим доступу : https://www.reviversoft.com/ru/file-extensions/p7s.

9. Проектування систем [Електронний ресурс]. – Режим доступу : https://goo.gl/CYbvyM.

10. Розробка інтерфейсу [Електронний ресурс]. – Режим доступу : https://apollo-8.ru/proektirovanie-interfeysov.

11. Особливість зберігання електронних документів [Електронний ресурс]. – Режим доступу : http://documentooborot.com/dokumentooborot/hranenie-elektronnyx-dokumentov.html.

12. Види сховищ електронних документів [Електронний ресурс]. – Режим доступу : https://studfiles.net/preview/5215241/page:14/.

13. Системи електронного документообігц [Електронний ресурс]. – Режим доступу : http://sed.reforms.in.ua/basic-page/pro-sed.

14. Хмарні сховища [Електронний ресурс]. – Режим доступу : http://fornote.net/2018/01/top-10-oblachny-h-hranilishh-2018-goda/.

15. Хмарні сховища [Електронний ресурс]. – Режим доступу : https://lifehacker.ru/best-cloud-storage-services/.

16. Модель збереження електронних документів у хмарі [Електронний ресурс]. – Режим доступу : https://goo.gl/5ykW76.

17. Модель збереження електронних документів у хмарі [Електронний ресурс]. – http://www.topobzor.com/obzor-10-oblachnyx-xranilishh-dannyx/.html.

18. Google Drive [Електронний ресурс]. – Режим доступу : https://goo.gl/Fube9u.

19. Microsoft OneDrive [Електронний ресурс]. – Режим доступу : https://goo.gl/kaHy3m.

20. DropBox [Електронний ресурс]. – Режим доступу : https://goo.gl/Xoz9Yw.

21. MEGA [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/Mega.

22. Особливості HotDocs системи [Електронний ресурс]. – Режим доступу : https://www.capterra.com/p/123091/HotDocs-Developer/.

23. Поняття блокчейну [Електронний ресурс]. – Режим доступу : https://habr.com/company/bitfury/blog/321474/.

24. Положення застосування ЕЦП в Україні [Електронний ресурс]. – Режим доступу : https://habr.com/company/bitfury/blog/321474/.

25. Auth 2.0 [Електронний ресурс]. – Режим доступу : https://oauth.net/2/.

26. Вузол навантаження [Електронний ресурс]. – Режим доступу : https://medium.com/coinmonks/the-bitcoin-network-6713cb8713d.

27. Васін Є.В. Програмний засіб виявляння SQL-ін’єкцій у вебзастосунках / Є.В. Васін, В.В. Цуркан // XI наукова конференція

магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК2018-2), 2018. – 4 с.

28. Ресурси IDEF0 [Електронний ресурс]. – Режим доступу : http://www.syque.com/quality_tools/tools/Tools19.htm.

29. Guid [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/GUID.

30. IDEF3 [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/IDEF3.

31. Звя'зки між процесами [Електронний ресурс]. – Режим доступу : https://www.cfin.ru/vernikov/idef/idef3.shtml.

32. Потоки даних [Електронний ресурс]. – Режим доступу : https://goo.gl/cMha2d.

33. Ангуляр застосунок даних [Електронний ресурс]. – Режим доступу : https://angular.io/guide/architecture.

34. Модулі сторінок [Електронний ресурс]. – Режим доступу : https://habr.com/post/351678/.

35. Архітектура веб-додатку на Ангуляр [Електронний ресурс]. – Режим доступу : https://angular.io/guide/architecture.

36. ASP.NET Core [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/ASP.NET_Core.

37. Angular CLI [Електронний ресурс]. – Режим доступу : https://goo.gl/Ju65Hx.

38. Переваги Angular [Електронний ресурс]. – Режим доступу : https://metanit.com/web/angular2/1.1.php.

39. TypeScript [Електронний ресурс]. – Режим доступу : https://www.quora.com/What-are-the-advantages-of-angular2.

40. MongoDb [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/MongoDB.

41. Переваги MongoDb [Електронний ресурс]. – Режим доступу : https://data-flair.training/blogs/advantages-of-mongodb/.

42. Trello [Електронний ресурс]. – Режим доступу : [https://ru.wikipedia.org/wiki/Trello.

43. Функціональні вимоги [Електронний ресурс]. – Режим доступу : https://studfiles.net/preview/2152457/page:4/.

44. Нефункціональні вимоги [Електронний ресурс]. – Режим доступу : https://habr.com/post/231961/.

45. Ручне тестування [Електронний ресурс]. – Режим доступу : https://qalight.com.ua/baza-znaniy/test-case/.

46. Тест-кейс [Електронний ресурс]. – Режим доступу : http://www.protesting.ru/testing/testcase.html.

47. Налаштування веб-сервера [Електронний ресурс]. – Режим доступу : https://metanit.com/sharp/aspnet5/2.7.php.

48. StoreDocApi.runtimeconfig [Електронний ресурс]. – Режим доступу : https://stackoverflow.com/questions/49642318/why-net-core-self-contained-app-depends-on-runtimeconfig-dev-json.

49. Дерево проблем [Електронний ресурс]. – Режим доступу : https://econ.wikireading.ru/44415.

50. Електронні договори [Електронний ресурс]. – Режим доступу : https://goo.gl/M7mVDd.

51. Сертифікат ключа [Електронний ресурс]. – Режим доступу : https://goo.gl/Wf8nyv.

52. RESTFull Api [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/REST.

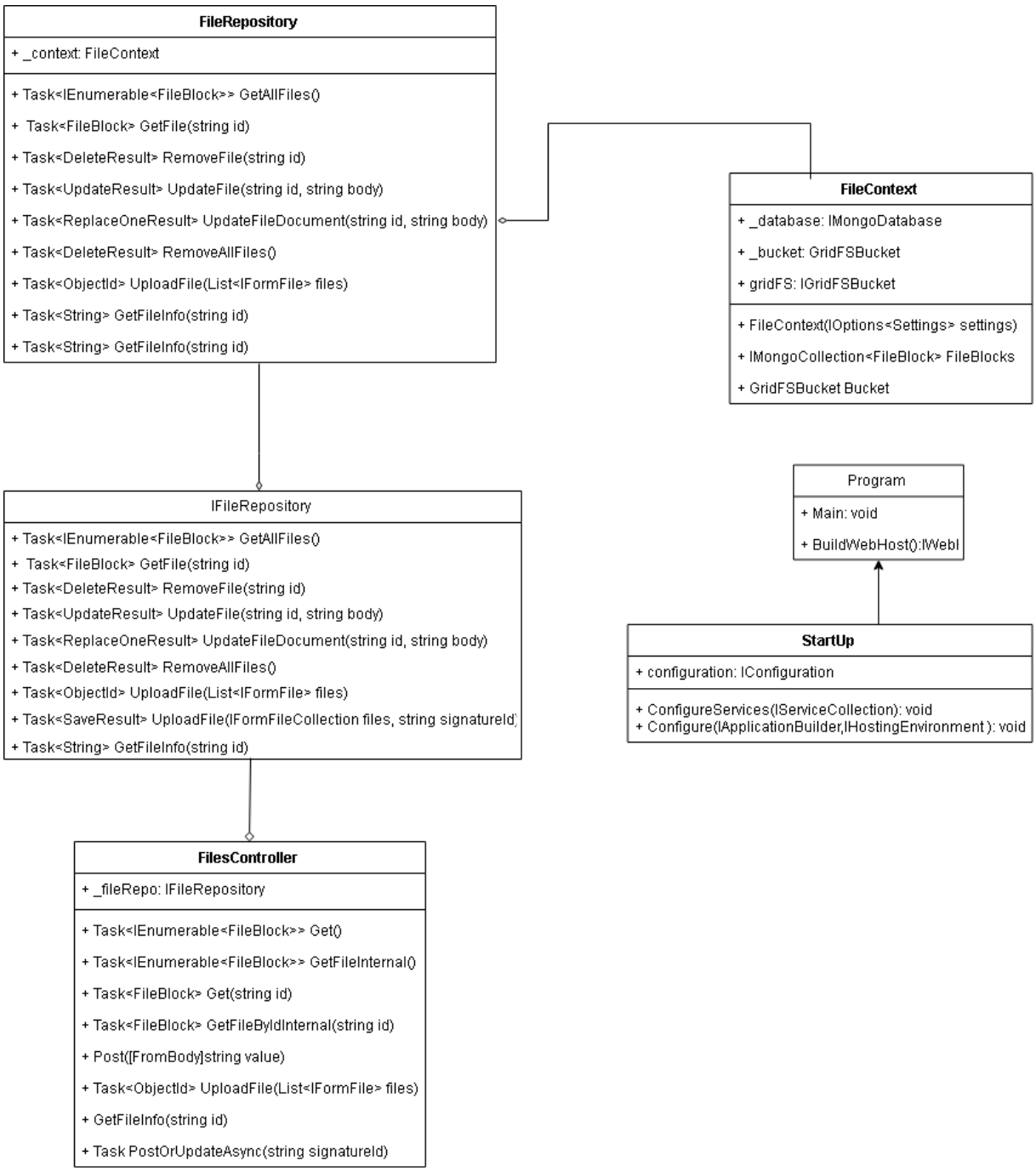
53. Абоненська плата [Електронний ресурс]. – Режим доступу : https://goo.gl/7kj3pB.

54. SWOT-анализ [Електронний ресурс]. – Режим доступу : http://powerbranding.ru/biznes-analiz/swot/.

55. Freemium Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/Freemium.

ДОДАТОК 1

Графічні матеріали



Діаграма класів
Васін К.В., КП-71мп

ДОДАТОК 2

Фрагмент тексту програми

FileController.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using MongoDB.Bson;
using StoreDocApi.Interfaces;
using StoreDocApi.Models;

namespace StoreDocApi.Controllers
{
    [Produces("application/json")]
    [Consumes("application/json", "multipart/form-data")]
    [Route("api/[controller]")]
    public class FilesController : ControllerBase
    {
        private readonly IFileRepository _fileRepo;

        public FilesController(IFileRepository repo)
        {
            _fileRepo = repo;
        }

        [NoCache]
        [HttpGet]
        public Task<IEnumerable<FileBlock>> Get()
        {
            return GetFileInternal();
        }

        private async Task<IEnumerable<FileBlock>> GetFileInternal()
        {
```

```
            return await _fileRepo.GetAllFiles();
        }

        // GET api/Files/5
        [HttpGet("{id}")]
        public Task<FileBlock> Get(string id)
        {
            return GetFileByIdInternal(id);
        }

        private async Task<FileBlock> GetFileByIdInternal(string id)
        {
            return await _fileRepo.GetFile(id) ?? new FileBlock();
        }

        // POST api/Files
        [HttpPost]
        public void Post([FromBody]string value)
        {
           // _fileRepo.AddFile(new FileBlock() { Body = value, CreatedOn =
DateTime.Now, UpdatedOn = DateTime.Now });
        }

        // PUT api/Files/5
        [HttpPut("{id}")]
        public void Put(string id, [FromBody]string value)
        {
            _fileRepo.UpdateFileDocument(id, value);
        }

        // DELETE api/Files/23243423
        [HttpDelete("{id}")]
        public void Delete(string id)
        {
            _fileRepo.RemoveFile(id);
        }

        // POST api/Files/uploadFile
        [HttpPost("uploadFile")]
        public async Task<ObjectId> UploadFile(List<IFormFile> files)
        {
            return await _fileRepo.UploadFile(files);
```

```
        }
        // GET api/Files/getFileInfo/d1we24ras41wr
        [HttpGet("getFileInfo/{id}")]
        public Task<String> GetFileInfo(string id)
        {
            return _fileRepo.GetFileInfo(id);
        }

        [HttpPost]
        [Route("upload")]
        public async Task PostOrUpdateAsync(string signatureId)
        {
            var files = HttpContext.Request.Form.Files;
            if (files.Count > 0)
            {
               await _fileRepo.UploadFile(files, signatureId);
            }
        }
    }
}
```

NoCacheAttribute.cs

```
using Microsoft.AspNetCore.Mvc.Filters;
using System;

namespace StoreDocApi.Controllers
{
    internal class NoCacheAttribute : ActionFilterAttribute
    {
        public override void OnResultExecuting(ResultExecutingContext
context)
        {
            context.HttpContext.Response.Headers["Cache-Control"] = "no-
cache, no-store, max-age=0";
            context.HttpContext.Response.Headers["Pragma"] = "no-cache";
            context.HttpContext.Response.Headers["Expires"] = "-1";

            base.OnResultExecuting(context);
        }
    }
}
```

FileContext.cs

```
using Microsoft.Extensions.Options;
using MongoDB.Driver;
using MongoDB.Driver.GridFS;
using StoreDocApi.Models;

namespace StoreDocApi.DbContext
{
    public class FileContext
    {
        private readonly IMongoDatabase _database = null;
        private readonly GridFSBucket _bucket = null;
        IGridFSBucket gridFS;   // файловое хранилище

        public FileContext(IOptions<Settings> settings)
        {
            var client = new MongoClient(settings.Value.ConnectionString);
            if (client != null)
            {
                _database = client.GetDatabase(settings.Value.Database);

                var gridFSBucketOptions = new GridFSBucketOptions()
                {
                    BucketName = "files",
                    ChunkSizeBytes = 1048576, // 1МБ
                };

                _bucket = new GridFSBucket(_database, gridFSBucketOptions);

            }
        }

        public IMongoCollection<FileBlock> FileBlocks
        {
            get
            {
                return _database.GetCollection<FileBlock>("FileBlock");
            }
        }
```

```
        public GridFSBucket Bucket
        {
            get
            {
                return _bucket;
            }
        }
    }
}
```

IFileRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using MongoDB.Driver;
using Microsoft.AspNetCore.Http;
using MongoDB.Bson;
using MongoDB.Driver.GridFS;
using StoreDocApi.Models;

namespace StoreDocApi.Interfaces
{
    public interface IFileRepository
    {
        Task<IEnumerable<FileBlock>> GetAllFiles();
        Task<FileBlock> GetFile(string id);
        Task AddFile(FileBlock item);
        Task<DeleteResult> RemoveFile(string id);

        Task<UpdateResult> UpdateFile(string id, string body);

        // demo interface - full document update
        Task<ReplaceOneResult> UpdateFileDocument(string id, string body);

        // should be used with high cautious, only in relation with demo
setup
        Task<DeleteResult> RemoveAllFiles();

        Task<ObjectId> UploadFile(List<IFormFile> files);
```

```
        Task<SaveResult> UploadFile(IFormFileCollection files, string
signatureId);
        Task<String> GetFileInfo(string id);
    }
}
```

FileBlocks.cs

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace StoreDocApi.Models
{
    public class FileBlock
    {
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }
        public string Hash { get; set; } = string.Empty;
        public string PrevBlockHash { get; set; } = string.Empty;
        public DateTime Date { get; set; } = DateTime.Now;
        public string UserId { get; set; } = string.Empty;
        public string FileName { get; set; } = string.Empty;
        public string FileId { get; set; }
        public string SignatureName { get; set; } = string.Empty;
        public string SignatureId { get; set; }
        public string SignatureUrl { get; set; }
        public string FileUrl { get; set; }

    }
}
```

IFileRepository.cs

```
using StoreDocApi.DbContext;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

using MongoDB.Bson;
using Microsoft.AspNetCore.Http;
using System.IO;
using Microsoft.Net.Http.Headers;
using MongoDB.Driver.GridFS;
using StoreDocApi.Models;
using StoreDocApi.Interfaces;
using System.Net.Http;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;
using Microsoft.AspNetCore.Cryptography.KeyDerivation;

namespace StoreDocApi.Repository
{
    public class FileRepository : IFileRepository
    {
        private readonly FileContext _context = null;

        public FileRepository(IOptions<Settings> settings)
        {
            _context = new FileContext(settings);
        }

        public async Task<IEnumerable<FileBlock>> GetAllFiles()
        {
            try
            {
                return await _context.FileBlocks.Find(_ =>
true).ToListAsync();
            }
            catch (Exception ex)
            {
                // log or manage the exception
                throw ex;
```

```
    }
}

public async Task<FileBlock> GetFile(string id)
{
    var filter = Builders<FileBlock>.Filter.Eq("Id", id);

    try
    {
        return await _context.FileBlocks
                        .Find(filter)
                        .FirstOrDefaultAsync();
    }
    catch (Exception ex)
    {
        // log or manage the exception
        throw ex;
    }
}

public async Task AddFile(FileBlock item)
{
    try
    {
        await _context.FileBlocks.InsertOneAsync(item);

    }
    catch (Exception ex)
    {
        // log or manage the exception
        throw ex;
    }
}

public async Task<DeleteResult> RemoveFile(string id)
{
    try
    {
        return await _context.FileBlocks.DeleteOneAsync(
             Builders<FileBlock>.Filter.Eq("Id", id));
    }
    catch (Exception ex)
```

```
            {
                // log or manage the exception
                throw ex;
            }
        }

        public async Task<UpdateResult> UpdateFile(string id, string body)
        {
            var filter = Builders<FileBlock>.Filter.Eq(s => s.Id, id);
            var update = Builders<FileBlock>.Update
                            .Set(s => s.FileName, body)
                            .CurrentDate(s => s.Date);

            try
            {
                return await _context.FileBlocks.UpdateOneAsync(filter,
update);
            }
            catch (Exception ex)
            {
                // log or manage the exception
                throw ex;
            }
        }

        public async Task<ReplaceOneResult> UpdateFile(string id, FileBlock
item)
        {
            try
            {
                return await _context.FileBlocks
                            .ReplaceOneAsync(n => n.Id.Equals(id)
                                            , item
                                            , new UpdateOptions { IsUpsert
= true });
            }
            catch (Exception ex)
            {
                // log or manage the exception
                throw ex;
            }
        }
```

```
        // Demo function - full document update
        public async Task<ReplaceOneResult> UpdateFileDocument(string id,
string body)
        {
            var item = await GetFile(id) ?? new FileBlock();
            //item.Body = body;
            //item.UpdatedOn = DateTime.Now;

            return await UpdateFile(id, item);
        }

        public async Task<DeleteResult> RemoveAllFiles()
        {
            try
            {
                return await _context.FileBlocks.DeleteManyAsync(new
BsonDocument());
            }
            catch (Exception ex)
            {
                // log or manage the exception
                throw ex;
            }
        }

        public async Task<ObjectId> UploadFile(List<IFormFile> files)
        {
            try
            {
                var file = files.FirstOrDefault();
                var stream = file.OpenReadStream();
                var filename = file.FileName;
                int signatureId = 10;

                var url =
$"https://test.ukey.net.ua:3020/api/v1/signatures/file/{signatureId}?mode=f
ull";
                using (HttpClient client = new HttpClient())
                {
                    client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Token", "w5n77v3GRLGq6RJMybpVeaPv6V7sogG3");
```

```
                    using (HttpResponseMessage res = await
client.GetAsync(url))
                    using (HttpContent content = res.Content)
                    {
                        string data = await content.ReadAsStringAsync();
                        if (data != null)
                        {
                            Console.WriteLine(data);
                        }
                    }
                }

                return await
_context.Bucket.UploadFromStreamAsync(filename, stream);
            }
            catch (Exception ex)
            {
                // log or manage the exception
                return new ObjectId(ex.ToString());
            }
        }

        public async Task<SaveResult> UploadFile(IFormFileCollection files,
string signatureId)
        {
            var result = new SaveResult()
            {
                UserId = "",
                FileId = "",
                Message = "Ошибка!"
            };

            var file = files.FirstOrDefault();
            var fileStream = file.OpenReadStream();
            var fileName = file.FileName;
            var baseUrl = $"https://test.ukey.net.ua:3020";
            var url = $"{baseUrl}/api/v1/signatures/file/{signatureId}";
            var signatureRequest = "";
            try
            {
                using (HttpClient client = new HttpClient())
                {
```

```
                    client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", "w5n77v3GRLGq6RJMybpVeaPv6V7sogG3");
                    using (HttpResponseMessage res = await
client.GetAsync(url))
                    {
                        using (HttpContent content = res.Content)
                        {
                            signatureRequest = await
content.ReadAsStringAsync();
                        }
                        if (signatureRequest != null)
                        {
                            (string userId, string signUrl, string fileUrl)
= GetSignData(JsonConvert.DeserializeObject<Sign>(signatureRequest));

                            using (HttpResponseMessage httpResult = await
client.GetAsync(baseUrl + signUrl))
                            {
                                using (HttpContent p7sFile =
httpResult.Content)
                                {
                                    var signatureStream = await
p7sFile.ReadAsStreamAsync();
                                    var actionResult =  await
StoreFiles(userId, fileStream, fileName, signatureStream,
$"{fileName}.ps7", signUrl, fileUrl);
                                    return actionResult;
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    result.Message = ex.Message;
                    return result;
                }

                return result;
            }
```

```
        private async Task<SaveResult> StoreFiles(string userId, Stream
docStream, string docName, Stream signatureStream, string signatureName,
string signUrl, string fileUrl)
        {
            ObjectId fileId = await
_context.Bucket.UploadFromStreamAsync(docName, docStream);
            ObjectId signatureId = await
_context.Bucket.UploadFromStreamAsync(signatureName, signatureStream);

            var block = new FileBlock()
            {
                FileName = docName,
                SignatureName = signatureName,
                Date = DateTime.Now,
                UserId = userId,
                FileId = fileId.ToString(),
                SignatureId = signatureId.ToString(),
                Id = fileId.ToString(),
                Hash = docName + userId,
                SignatureUrl = signUrl,
                FileUrl = fileUrl
            };
            //block.Hash = CreateHash(docName + userId);
            var lastOneBlock = GetLastOneBlock();
            if (lastOneBlock != null)
            {
                block.PrevBlockHash = lastOneBlock.Hash;
            }

            await _context.FileBlocks.InsertOneAsync(block);
            var result = new SaveResult()
            {
                UserId = userId,
                FileId = fileId.ToString(),
                Message = "Файл успішно збережений!"
            };
            return result;
        }

        string CreateHash(string str)
        {
            byte[] salt = new byte[128 / 8];
```

```
            string hash = Convert.ToBase64String(KeyDerivation.Pbkdf2(
               password: str,
               salt: salt,
               prf: KeyDerivationPrf.HMACSHA1,
               iterationCount: 10000,
               numBytesRequested: 256 / 8));

            return hash;
        }

        private (string userId, string url, string fileUrl)
GetSignData(Sign sign)
        {
            string signUrl = sign.fileSignatures.signatures[0].url;
            string userId = sign.userId;
            string fileUrl = sign.fileSignatures.signatures[0].url;
            return (userId, signUrl, fileUrl);
        }

        FileBlock GetLastOneBlock()
        {
            return _context.FileBlocks.AsQueryable().Where(l =>
l.Hash.Length > 0).AsQueryable().OrderByDescending(l =>
l.Date).FirstOrDefault();
        }

        public async Task<String> GetFileInfo(string id)
        {
            GridFSFileInfo info = null;
            var objectId = new ObjectId(id);
            try
            {
                using (var stream = await
_context.Bucket.OpenDownloadStreamAsync(objectId))
                {
                    info = stream.FileInfo;
                }
                return info.Filename;
            }
            catch (Exception)
            {
                return "Not Found";
```

```
            }
        }
    }

    public class Sign
    {
        public Signature fileSignatures { get; set; }
        public string userId { get; set; }
    }

    public class Signature
    {
        public FileDetails[] signatures { get; set; }
        public Payload filePayload { get; set; }

    }

    public class Payload
    {
        public FileDetails[] payload { get; set; }
    }

    public class FileDetails
    {
        public string url { get; set; }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace StoreDocApi
{
```

```
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}
```

Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using StoreDocApi.DbContext;
using StoreDocApi.Interfaces;
using StoreDocApi.Models;
using StoreDocApi.Repository;

namespace StoreDocApi
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }
```

```
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            // Add service and create Policy with options
            services.AddCors(options =>
            {
                options.AddPolicy("CorsPolicy",
                    builder => builder.AllowAnyOrigin()
                    .AllowAnyMethod()
                    .AllowAnyHeader()
                    .AllowCredentials());
            });
            services.AddSingleton<FileContext>();
            services.AddCors(options =>
            {
                options.AddPolicy("AllowAll",
                    builder =>
                    {
                        builder
                        .AllowAnyOrigin()
                        .AllowAnyMethod()
                        .AllowAnyHeader()
                        .AllowCredentials();
                    });
            });
            // Add framework services.
            services.AddMvc();
            services.Configure<Settings>(options =>
            {
                options.ConnectionString =
Configuration.GetSection("MongoConnection:ConnectionString").Value;
                options.Database =
Configuration.GetSection("MongoConnection:Database").Value;
            });

            services.AddTransient<IFileRepository, FileRepository>();
        }
```

```
        // This method gets called by the runtime. Use this method to
configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.UseCors("AllowAll");

            app.UseMvc();
        }
    }
}
```

SaveResult.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace StoreDocApi.Models
{
    public class SaveResult
    {
        public string FileId { get; set; }
        public string UserId { get; set; }
        public string Message { get; set; }
    }
}
```

Settings.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace StoreDocApi.Models
```

```
{
    public class Settings
    {
        public string ConnectionString;
        public string Database;
    }
```

auth.component.cs

```
import { Component, OnInit, HostListener } from '@angular/core';
import { UKeyConstants } from '../../../assets/ukey-plugin/js/constants';
import { UploadEvent, UploadFile, FileSystemFileEntry,
FileSystemDirectoryEntry } from 'ngx-file-drop';
import { UKeyService } from '../services/ukey.service';
import { Observable, interval, Subject, timer } from 'rxjs';
import { flatMap, takeWhile, switchMap, map, startWith } from
'rxjs/operators';
import { SuccessResponce } from '../models/succesResonce';
import { AlertService } from '../services/alert.service';

@Component({
  selector: 'app-add-doc',
  templateUrl: './add-doc.component.html',
  styleUrls: ['./add-doc.component.scss']
})
export class AddDocComponent implements OnInit {

  file: File
  token: SuccessResponce
  signatureId: string
  tokenStr: string = localStorage.getItem('UKeytoken');
  flow: any = {
    isFileAdded: false,
    isFileSigned: false,
    isFileSent: false
  }

  constructor(private UKey: UKeyService, private alert: AlertService) { }
```

```
  ngOnInit() {
    this.timer$ = this.reset$.pipe(
      startWith(0),
      switchMap(() => timer(0, 1000))
    );
  }
  private reset$ = new Subject();
  timer$: Observable<any>;

  public files: UploadFile[] = [];

  public dropped(event: UploadEvent) {
    this.files = event.files;
    for (const droppedFile of event.files) {

      // Is it a file?
      if (droppedFile.fileEntry.isFile) {
        const fileEntry = droppedFile.fileEntry as FileSystemFileEntry;
        fileEntry.file((file: File) => {

          // Here you can access the real file
          console.log(droppedFile.relativePath, file);
          this.file = file
          this.alert.success(`Файл завантажено`);
          this.flow.isFileAdded = true;
          this.flow.isFileSigned = false
          this.flow.isFileSent = false
        });
      } else {
        // It was a directory (empty directories are added, otherwise only
files)
        const fileEntry = droppedFile.fileEntry as
FileSystemDirectoryEntry;
        console.log(droppedFile.relativePath, fileEntry);
      }
    }
  }

  signWithUKey() {
    let id;
    this.UKey.mobileAuth().subscribe(
      data => {
```

```
      interval(2000)
        .pipe(
          flatMap(() => this.UKey.checkMobileAuth(data.id)),
          takeWhile(data => {
            debugger
            if (data.token) {
              localStorage.setItem("UKeytoken", JSON.stringify(data))
              this.tokenStr = JSON.stringify(data)
              return false
            }
            else {
              return true
            }
          }))
        .subscribe(result => console.log(result));
    },
    error => { debugger }
  );
}

signFileWithUKey() {
  debugger
  let token = JSON.parse(localStorage.getItem("UKeytoken"))
  this.UKey.signFile(token.token.access_token, this.file, this.file.name)
    .subscribe(
      data => {
        debugger
        interval(2000)
          .pipe(
            flatMap(() => this.UKey.checkSignatureId(data.requestId)),
            takeWhile(data => {
              if(data.status == "PENDING"){
                return true;
              }
              if(data.status == "SIGNED"){
                this.signatureId = data.resultId;
                this.flow.isFileSigned = true
                this.UKey.triggerLoading(false);
                this.alert.success(`Файл підписано`);
                return false;
              }
```

```
              }))
            .subscribe(result => console.log(result));
        })
    error => { }
  }

  sendFile() {
    this.UKey.sendFile(this.file, this.file.name,
this.signatureId).subscribe(
      data => {
        this.flow.isFileSent = true
        this.alert.success(`Файл відправлено`);
        debugger
      },
      error => {
        debugger
      }
    )
  }

}
import { Component, OnInit, HostListener } from '@angular/core';
import { UKeyConstants } from '../../../assets/ukey-plugin/js/constants';
import { UploadEvent, UploadFile, FileSystemFileEntry,
FileSystemDirectoryEntry } from 'ngx-file-drop';
import { UKeyService } from '../services/ukey.service';
import { Observable, interval, Subject, timer } from 'rxjs';
import { flatMap, takeWhile, switchMap, map, startWith } from
'rxjs/operators';
import { SuccessResponce } from '../models/succesResonce';
import { AlertService } from '../services/alert.service';

@Component({
  selector: 'app-add-doc',
  templateUrl: './add-doc.component.html',
  styleUrls: ['./add-doc.component.scss']
})
export class AddDocComponent implements OnInit {

  file: File
  token: SuccessResponce
  signatureId: string
```

```
  tokenStr: string = localStorage.getItem('UKeytoken');
  flow: any = {
    isFileAdded: false,
    isFileSigned: false,
    isFileSent: false
  }

  constructor(private UKey: UKeyService, private alert: AlertService) { }

  ngOnInit() {
    this.timer$ = this.reset$.pipe(
      startWith(0),
      switchMap(() => timer(0, 1000))
    );
  }
  private reset$ = new Subject();
  timer$: Observable<any>;

  public files: UploadFile[] = [];

  public dropped(event: UploadEvent) {
    this.files = event.files;
    for (const droppedFile of event.files) {

      // Is it a file?
      if (droppedFile.fileEntry.isFile) {
        const fileEntry = droppedFile.fileEntry as FileSystemFileEntry;
        fileEntry.file((file: File) => {

          // Here you can access the real file
          console.log(droppedFile.relativePath, file);
          this.file = file
          this.alert.success(`Файл завантажено`);
          this.flow.isFileAdded = true;
          this.flow.isFileSigned = false
          this.flow.isFileSent = false
        });
      } else {
        // It was a directory (empty directories are added, otherwise only
files)
        const fileEntry = droppedFile.fileEntry as
FileSystemDirectoryEntry;
```

```
      console.log(droppedFile.relativePath, fileEntry);
    }
  }
}

signWithUKey() {
  let id;
  this.UKey.mobileAuth().subscribe(
    data => {

      interval(2000)
        .pipe(
          flatMap(() => this.UKey.checkMobileAuth(data.id)),
          takeWhile(data => {
            debugger
            if (data.token) {
              localStorage.setItem("UKeytoken", JSON.stringify(data))
              this.tokenStr = JSON.stringify(data)
              return false
            }
            else {
              return true
            }
          }))
        .subscribe(result => console.log(result));
    },
    error => { debugger }
  );
}

signFileWithUKey() {
  debugger
  let token = JSON.parse(localStorage.getItem("UKeytoken"))
  this.UKey.signFile(token.token.access_token, this.file, this.file.name)
    .subscribe(
      data => {
        debugger
        interval(2000)
          .pipe(
            flatMap(() => this.UKey.checkSignatureId(data.requestId)),
            takeWhile(data => {
              if(data.status == "PENDING"){
```

```
                return true;
              }
              if(data.status == "SIGNED"){
                this.signatureId = data.resultId;
                this.flow.isFileSigned = true
                this.UKey.triggerLoading(false);
                this.alert.success(`Файл підписано`);
                return false;
              }
            }))
          .subscribe(result => console.log(result));
      })
    error => { }
  }

  sendFile() {
    this.UKey.sendFile(this.file, this.file.name,
this.signatureId).subscribe(
      data => {
        this.flow.isFileSent = true
        this.alert.success(`Файл відправлено`);
        debugger
      },
      error => {
        debugger
      }
    )
  }

}
import { Component, OnInit } from '@angular/core';
import { UKeyService } from '../services/ukey.service';
import { Block } from '../models/block';
import {saveAs as importedSaveAs} from "file-saver";

@Component({
  selector: 'app-doc-list',
  templateUrl: './doc-list.component.html',
  styleUrls: ['./doc-list.component.scss']
})
export class DocListComponent implements OnInit {
```

```
  blocks: Array<Block>;

  constructor(private UKey: UKeyService) { }

  ngOnInit() {
    this.reload();
  }


  reload(){
    this.UKey.getFiles().subscribe(data => this.blocks = data);
  }

  loadFile(url: string, fileName: string){
debugger
    this.UKey.loadFile(url).subscribe(
      data =>{
        let blob = new Blob([data], { type: "application/octet-stream"});
        importedSaveAs(blob, fileName);
      }
    )


  }



}
```